Universidad Autónoma de Madrid

Escuela Politécnica Superior

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

APLICACIÓN ANDROID PARA EL APRENDIZAJE DE MÉTODOS DE RESOLUCIÓN AVANZADOS DEL CUBO DE RUBIK

Javier Sánchez Alonso
Tutor: Gonzalo Martínez Muñoz

Julio 2015

APLICACIÓN ANDROID PARA EL APRENDIZAJE DE MÉTODOS DE RESOLUCIÓN AVANZADOS DEL CUBO DE RUBIK

Autor: Javier Sánchez Alonso
Tutor: Gonzalo Martínez Muñoz

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Julio 2015

Agradecimientos

A Gonzalo, por aceptar mi propuesta de Trabajo de Fin de Grado y permitirme así poder disfrutar de la elaboración de este proyecto.

A todos aquellos que participaron en la realización de la plantilla LaTeX sobre la que se ha elaborado este documento. En especial a Juan Sidrach, por su esfuerzo en divulgarla.

A todos los que se han interesado por mi aplicación y a los que me han pedido tenerla en sus móviles. Me he nutrido de su ilusión para mejorar.

A Jaime y Álvaro, que han aportado color.

A Gabriela, cuya paciencia, habilidad para escuchar y recomendaciones me han servido para mejorar el diseño.

A todos aquellos que disfrutan con el cubo de Rubik y no le quitan las pegatinas. Esta aplicación es para vosotros.

Abstract

The Rubik's cube is one of the most popular puzzles of all time. It is highly difficult to solve it, so lots of people end up following internet guides. Besides guides for novel people, there are also other guides to learn how to solve the Rubik's cube in a much faster way. They are used by those whose intention is to beat personal records or even compete internationally. Most of the resolution methods require memorizing step by step some rotation sequences —called «algorithms»—. The problem is that some methods contain dozens of algorithms to be learned.

Each algorithm can be described as a string of characters that defines the rotations it is composed, i.e.: «l2' U' z' U R2 U' L U R2 z' L». The guides use this notation to display the algorithms. However, for users not used to the notation it is tedious to interprete.

The aim of this project is to develop an app to learn the resolution methods in an easy way. This have been achieved in two ways. First of them has been developing an Android application, so that the mobility of smartphones and tablets is used to allow the user to consult, at any time, anywhere, the algorithms he or she wants to learn. The second way has been building a Rubik's cube in three dimensions, so it allows to run any algorithm.

The application features an elegant user interface that follows the design guidelines recommended for Android by Google, and it provides a positive user experience. Moreover, the structure of the application is scalable so that, if it is necessary to include new resolution methods or algorithms, there is no need of making changes in it. Finally, it includes extra functionality as the ability to save backups of application data in the cloud or using different color schemes to display the three-dimensional Rubik's cube.

Key words — Rubik's cube, Android, Material design, OpenGL

Resumen

El cubo de Rubik es uno de los rompecabezas más populares de todos los tiempos. La dificultad de resolverlo es alta, por lo que mucha gente termina acudiendo a guías existentes en internet. Además de guías para gente novel, también existen otras destinadas a resolver el cubo de Rubik de una manera mucho más rápida. Éstas son utilizadas por aquellos cuya intención es batir récords personales o incluso competir a nivel internacional. La mayoría de métodos de resolución requieren que se memoricen paso a paso secuencias de giros —llamadas «algoritmos»—. El problema de algunos métodos es que contienen decenas de estos algoritmos.

Cada algoritmo puede expresarse a través de una cadena de caracteres que define los giros que lo componen, por ejemplo: «l2' U' z' U R2 U' L U R2 z' L». Las guías hacen uso de esta notación para mostrar los algoritmos. No obstante, para el usuario no habituado a ella, tener que interpretarla es una labor tediosa.

El objetivo de este proyecto es construir una aplicación que facilite el aprendizaje de los diferentes métodos de resolución. Esto se ha conseguido de dos formas. La primera ha sido desarrollando la aplicación para Android, de tal manera que se aprovecha la movilidad de los smartphones y tabletas para que el usuario pueda consultar en cualquier momento y lugar aquellos algoritmos que quiera aprender. La segunda ha sido construyendo un cubo de Rubik en tres dimensiones que permite ejecutar cualquier algoritmo que se le indique.

La aplicación cuenta con una interfaz gráfica cuidada, que sigue las líneas de diseño recomendadas por Google para Android y que proporciona una experiencia de usuario positiva. Además, la estructura de la aplicación es escalable de forma que, si se quieren incorporar nuevos métodos de resolución y algoritmos, no es necesario realizar cambios en ella. Por último, incluye funcionalidad extra como la posibilidad de guardar en la nube copias de seguridad de los datos de la aplicación o de emplear distintos esquemas de colores para mostrar el cubo de Rubik tridimensional.

Palabras clave — cubo de Rubik, Android, Material design, OpenGL

Glosario

3OP *3-Cycle Orientation Permutation*, «3-ciclo orientación permutación». Método de resolución del cubo de Rubik a ciegas. 20, 57

Algoritmo Sucesión de rotaciones del cubo de Rubik destinada a resolver un estado. V, 2, 5, 7, 9, 10, 13–15, 17, 19–21, 23, 26, 27, 30, 36, 37, 45, 47, 51, 52

API *Application Programming Interface*, «interfaz de programación de aplicaciones». Conjunto de subrutinas, funciones y procedimientos que ofrece una biblioteca para ser utilizado por otro software como capa de abstracción. 2, 17, 33, 42, 47

CFOP *Cross, F2L, OLL, PLL*, «Cruz, F2L, OLL, PLL». El más extendido de los métodos de resolución rápida del cubo de Rubik. También llamado «método Fridrich» por ser Jessica Fridrich quien lo popularizó. 5, 6, 9, 10, 20, 57

Cross «Cruz». Paso cuyo fin es completar las aristas de la primera cara. 6

Estado Disposición específica de la totalidad o parte de los cubitos que forman el cubo de Rubik. Los estados modelizan las diferentes posibilidades en las que un usuario puede tener su cubo de Rubik. 5–11, 19–21, 23, 26, 27, 29, 30, 36–38, 44, 45, 47

F2L *First Two Layers*, «dos primeras capas». Paso cuyo fin es completar las dos primeras capas -la inferior y la del medio-. 6, 76

IDE *Integrated Development Environment*, «entorno de desarrollo integrado». Aplicación que facilita al programador el desarrollo de software. 15, 50

JSON *JavaScript Object Notation*, «notación de objetos de JavaScript». Formato ligero para el intercambio de datos. 35–38, 47

Material design Conjunto de pautas de diseño establecidas por Google y recomendadas para su empleo en las aplicaciones Android. 16, 20, 28, 51

Método de resolución Forma de resolver un cubo de Rubik, ya sea de manera parcial o total. Cada método está dividido en pasos. V, 1, 2, 5–10, 13, 15, 19–21, 23, 25, 37, 53, 57, 58

OLL *Orientation of the Last Layer*, «orientación de la última capa». Paso cuyo fin es orientar la última capa. 6, 9, 10, 26, 76

OpenGL ES Variante simplificada de la API gráfica OpenGL, empleada en este Trabajo de Fin de Grado para crear el cubo de Rubik en tres dimensiones. 2, 41

Paso Etapas en las que se divide un método, empleadas para organizar la resolución. Cada paso agrupa un conjunto de estados. 5–7, 9–11, 19–21, 23, 25, 26, 29, 30, 36–38, 45

PLL *Permutation of the Last Layer*, «permutación de la última capa». Paso cuyo fin es permutar la última capa. 6, 9, 10, 76

RGB *Red, Green, Blue*, «rojo, verde, azul». Composición del color en términos de la intensidad de los colores primarios de la luz. 28, 37, 41

SDK *Software Development Kit*, «kit de desarrollo de software». Conjunto de herramientas de desarrollo software que permiten crear aplicaciones para un sistema concreto. 15–17, 39, 47

XML *eXtensible Markup Language*, «lenguaje de marcas extensible». Lenguaje de marcas utilizado para almacenar datos en forma legible. 38, 39, 45

Índice general

Índice de figuras

1 Introducción

El presente Trabajo de Fin de Grado tiene como propósito el desarrollo de una aplicación Android para el aprendizaje de métodos de resolución del cubo de Rubik.

1.1. Motivación

El cubo de Rubik es un rompecabezas inventado por el húngaro Ernő Rubik en 1974 y uno de los juguetes más populares de todos los tiempos.

Aprender a resolver el cubo de Rubik es una tarea difícil que lleva tiempo, lo que hace que la mayoría de las personas que quieran solucionarlo se sirvan de guías.

Los más entusiastas de este rompecabezas no se limitan a resolverlo, sino que van más allá y su principal objetivo es hacerlo en un tiempo récord. Otros se enorgullecen de haberlo resuelto a ciegas, con una sola mano, con los mínimos giros posibles, etc. Es tal el interés originado en torno al cubo de Rubik y sus variantes, que existe una asociación, la World Cube Association [1], que organiza campeonatos a nivel mundial.

El abanico de maneras y propósitos de solucionar el cubo ha llevado a que se desarrollen métodos diferentes adaptados a cada forma de resolución. Las miles de personas que se adentran en este mundo se sirven de guías existentes en internet. En ellas se muestran las centenas de series de giros que deben ser aprendidas de memoria a través de una práctica continua. La mejor forma de exponerlas es mediante vídeos o applets en Java de modelos tridimensionales del cubo de Rubik, siendo esta última opción la más empleada por su posibilidad de representación automática de rotaciones.

Así pues, hoy día, es necesario contar con un ordenador con conexión a internet durante

este largo camino de aprendizaje. La principal motivación de este Trabajo de Fin de Grado es liberar a las personas de esta importante limitación y dotarlas de una nueva herramienta de aprendizaje que aproveche la movilidad de los smartphones.

1.2. Objetivos

Los objetivos principales de este Trabajo de Fin de Grado son los siguientes:

- Construir un cubo de Rubik tridimensional capaz de ejecutar rotaciones. Se realizará con OpenGL ES.
- Crear una interfaz a la vez funcional y usable para que el usuario sea capaz de aprender rápidamente a navegar por la aplicación y lo haga de una manera ágil y cómoda.
- Incorporar los métodos de resolución del cubo de Rubik más extendidos. Se tendrán en cuenta el método para principiantes y los métodos más conocidos de resolución rápida y a ciegas.
- Dar la posibilidad al usuario de crear sus propias secuencias de rotaciones y editar las incorporadas por defecto en la aplicación.
- Crear un sistema de copia de seguridad online de los algoritmos existentes en la aplicación. Para ello se empleará la API de Dropbox.

1.3. Estructura del documento

El resto del documento tiene la siguiente estructura:

- En el capítulo 2 se realiza un análisis del estado del arte. En él se explican los principios de los métodos de resolución del cubo de Rubik, se examinan las soluciones existentes que facilitan su aprendizaje y se extraen conclusiones sobre la necesidad del presente Trabajo de Fin de Grado.
- En el capítulo 3 se define el alcance del Trabajo de Fin de Grado, la metodología seguida durante la vida del proyecto y las herramientas empleadas para su elaboración.
- En el capítulo 4 se enumeran los requisitos funcionales y no funcionales de la aplicación.
- En el capítulo 5 se describe el diseño de la aplicación, que incluye: el patrón de arquitectura, la base de datos y la interfaz gráfica.

- En el capítulo 6 se detalla cómo se han implementado los diferentes módulos del proyecto.
- En el capítulo 7 se describen las pruebas realizadas para verificar y validar la aplicación.
- En el capítulo 8 se especifica qué mantenimiento se va a llevar a cabo.
- En el capítulo 9 se exponen las conclusiones sacadas tras la conclusión del trabajo realizado.
- En el capítulo 10 se detallan las líneas de trabajo futuro a seguir tras el fin del Trabajo de Fin de Grado.

2

Estado del arte

La complejidad del cubo de Rubik lleva a que las personas que quieren resolver por vez primera este rompecabezas lo hagan a través de un método mecánico relativamente sencillo de memorizar. Estos métodos se caracterizan por estar compuestos por pocos algoritmos. De manera contraria, los métodos de resolución rápida —usados por todos aquellos que compiten a nivel mundial— se caracterizan por su complejidad, su gran cantidad de algoritmos y su eficacia.

En esta sección primero se explica en qué consiste un método de resolución, pues su importancia a lo largo del documento es máxima. Después se analizan las soluciones existentes en forma de applets de Java. Por último se estudian aplicaciones Android con un objetivo similar al pretendido por este Trabajo de Fin de Grado.

2.1. Resolución del cubo de Rubik

Los métodos de resolución del cubo de Rubik están compuestos por pasos que llevan el cubo de un estado a otro. Mientras que el estado final de un paso es siempre el mismo, el estado inicial puede variar. Por tanto, cada paso contiene a su vez una serie de estados iniciales posibles. Por último, cada estado se puede resolver ejecutando diferentes secuencias de giros, llamadas «algoritmos».

A modo de ejemplo que acompañe la explicación de cada término, tomaremos en esta sección el método CFOP, que es uno de los más extendidos entre aquellas personas que quieren batir récords de velocidad.

2.1.1. Métodos de resolución

Un método de resolución del cubo de Rubik es una forma de conseguir resolver parcial o totalmente el cubo. Totalmente si se parte de un cubo cualquiera y se termina resolviéndolo completamente. De manera parcial si los estados inicial y final no tienen por qué ser, respectivamente, un cubo deshecho aleatoriamente y un cubo resuelto completamente.

El método CFOP es un método de resolución total del cubo de Rubik.

2.1.2. Pasos

Los métodos se dividen en pasos. Cada paso tiene como objetivo llevar un cubo de un determinado estado a otro.

El nombre del método CFOP viene dado por sus pasos: Cross, F2L, OLL y PLL. Estos pasos se muestran en la figura 2.1 y se explican a continuación:

1. Cross (cruz). Se parte de un cubo deshecho cualquiera y se consigue tener las aristas de la primera capa bien colocadas.

2. F2L (*First Two Layers*, dos primeras capas). Se parte de un cubo con la cruz de la primera capa bien posicionada y se consigue tener las dos primeras capas resueltas.

3. OLL (*Orientation of the Last Layer*, orientación de la última capa). Se parte de un cubo con las dos primeras capas resueltas y se consigue orientar todas las piezas de la última capa, es decir, se consigue que la cara superior esté resuelta. Ver apéndice B para entender la distinción entre «cara» y «capa».

4. PLL (*Permutation of the Last Layer*, permutación de la última capa). Se parte de un cubo con las dos primeras capas resueltas y las piezas de la última capa bien orientadas y se consigue resolver el cubo de Rubik completamente.

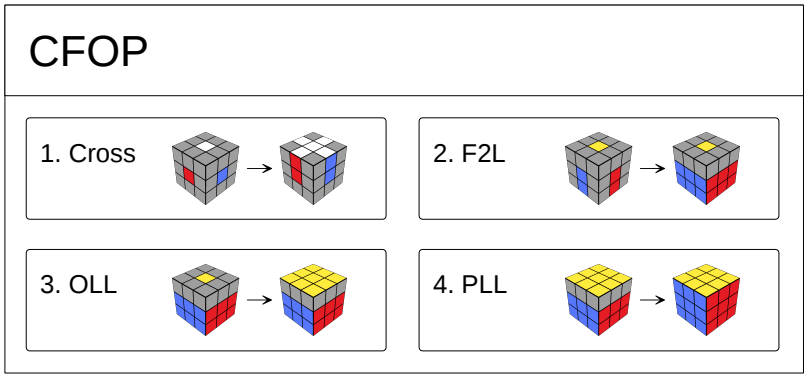

Figura 2.1: Pasos del método de resolución CFOP. En el paso *cross*, el cubo se encuentra girado 180º para poder ver la cara inferior, la de centro blanco.

2.1.3. Estados

La similitud entre el antes y el después de cualquier paso es que hay ciertas piezas —llamadas «cubitos»— que no se alteran. La diferencia es que hay otras que sí lo hacen. Un estado es la definición de la colocación de todas estas piezas. Como la disposición de los cubitos que importan en un paso es variable, para saber resolver tal paso hay que saber resolver cada uno de los estados posibles que pueden aparecer. Empleamos la expresión «resolver un estado» para decir que, tras una serie de giros, el paso al que pertenece el estado ha finalizado.

A la hora de resolver un estado, hay muchas pegatinas —caras de un cubito— cuyo color es irrelevante. Para facilitar la identificación rápida de los estados, estas pegatinas se pintan con un color neutro. A modo de ejemplo, en la figura 2.2, resolver el estado consiste en trasladar la arista azul y roja de la capa superior a la capa intermedia. Los cubitos de las dos primeras capas deben mantenerse en su sitio, de ahí que aparezcan pintados tanto en el estado inicial como en el final. Sin embargo, no importa que el resto de cubitos se descoloquen, son irrelevantes a la hora de resolver el estado; por ello, su color es el neutro, representado con gris.

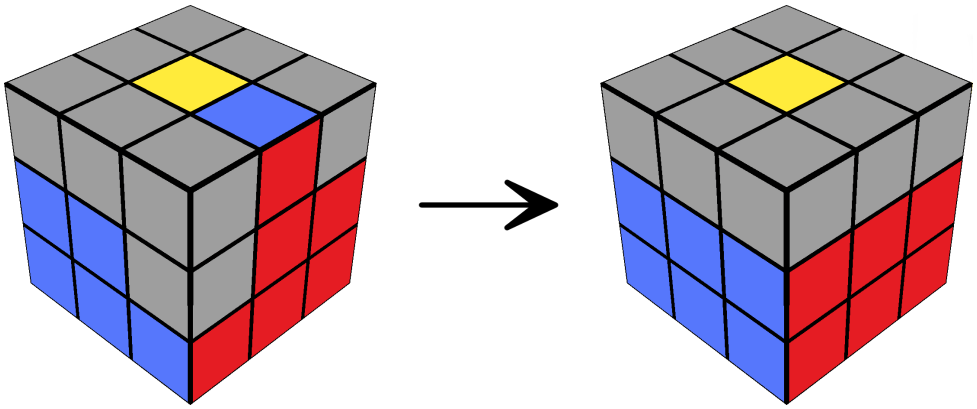

Figura 2.2: A la izquierda, un determinado estado inicial del paso F2L. Tras resolverse, obtenemos el estado final del paso.

2.1.4. Algoritmos

Un algoritmo es una secuencia ordenada de rotaciones del cubo de Rubik. Un estado se puede resolver con uno o más algoritmos.

En el apéndice B se detallan todos los tipos de rotaciones posibles.

2.2. Applets de Java

Muchas de las páginas web que detallan métodos de resolución se sirven de applets escritas en Java para mostrar cubos de Rubik tridimensionales que ejecutan algoritmos, es

decir, que giran según la secuencia de rotaciones que se le indique. Estas applets facilitan enormemente la labor de construcción de las guías de aprendizaje, pues con sencillez se pueden mostrar las decenas de estados que componen un método, así como la visualización de su resolución. Uno de los más usados es el construido por Werner Randelshofer [2], mostrado en la figura 2.3.

El problema de estas piezas de software es que actualmente la mayoría de los navegadores dificultan su ejecución por temas de seguridad, y los navegadores de los dispositivos móviles no les dan siquiera soporte.

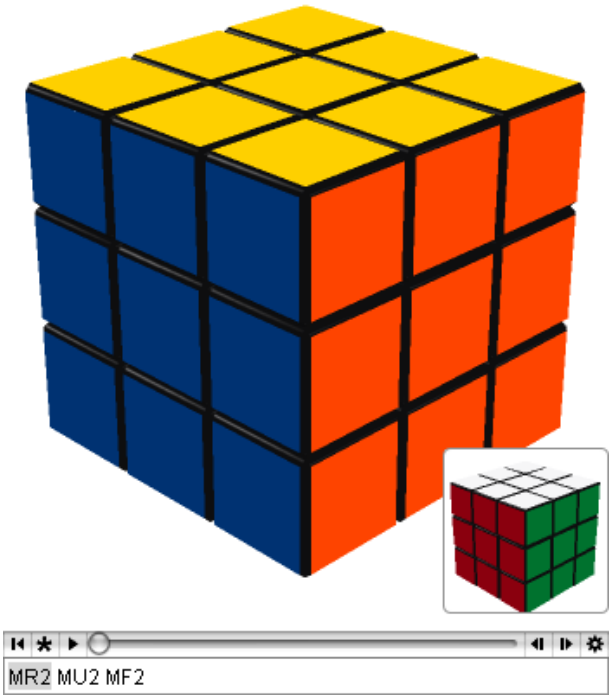

Figura 2.3: Applet construida por Werner Randelshofer

2.3. Aplicaciones similares

Además de las applets de Java, existen otro tipo de soluciones para el aprendizaje de métodos de resolución del cubo de Rubik, pero la mayoría de ellas requieren el uso de un ordenador con conexión a Internet. Dado que este Trabajo de Fin de Grado pretende eliminar dicha barrera y dotar al usuario de una aplicación Android offline que le permita aprender allá donde esté, en este apartado se estudian únicamente las aplicaciones existentes para este sistema operativo.

Las aplicaciones Android que se encuentran en Google Play relacionadas con el cubo de Rubik se pueden catalogar en tres tipos:

- Juegos en los que el usuario debe resolver el cubo de Rubik.

- Aplicaciones que sirven de guía para resolver el cubo con un método adecuado a gente novel.
- Aplicaciones que recopilan algoritmos de métodos de resolución rápida empleados por expertos.

El presente Trabajo de Fin de Grado se aleja de la primera opción y se acerca al propósito de las últimas.

Se han analizado una decena de aplicaciones de estas características. A continuación se detallan los aspectos positivos y negativos de las más destacadas.

2.3.1. *Cubo de Rubik Algoritmos y Más*

Desarrollada por JQR APPS, ha sido instalada entre 100.000 y 500.000 veces [3].

Aspectos positivos:

- Contiene algunos algoritmos para los cubos 4x4x4, 5x5x5 y 6x6x6.
- Incorpora un contador para registrar tiempos de resolución.
- Muestra enlaces a vídeos de YouTube y a récords mundiales.

Aspectos negativos:

- La aplicación se ve lastrada por la ingente cantidad de publicidad que muestra.
- La interfaz gráfica no cumple las líneas de diseño aconsejadas por Google.
- Únicamente se proporciona el método CFOP.
- Los estados de los pasos OLL y PLL solamente muestran un algoritmo cada uno.
- El usuario no puede modificar los algoritmos.
- Los algoritmos vienen dados únicamente por su definición escrita.

2.3.2. *Rubik's Cube Fridrich Solver*

Desarrollada por Bodor Gergely, ha sido instalada entre 50.000 y 100.000 veces [4].

Aspectos positivos:

- Permite al usuario introducir manualmente un estado del cubo de Rubik y da los pasos y algoritmos del método CFOP para resolver el cubo completamente.

- La aplicación es gratuita y no tiene publicidad.

Aspectos negativos:

- La interfaz gráfica no cumple las líneas de diseño aconsejadas por Google.
- Únicamente se proporciona el método CFOP.
- Los estados de los pasos OLL y PLL solamente muestran un algoritmo cada uno.
- El usuario no puede modificar los algoritmos.
- Los algoritmos vienen dados únicamente por su definición escrita.

2.3.3. *Rubik's Cube OLL/PLL Trainer*

Desarrollada por Dennis Hafemann, ha sido instalada entre 100 y 500 veces [5].

Aspectos positivos:

- La interfaz gráfica sigue las líneas de diseño aconsejadas por Google.
- Permite al usuario indicar qué estados sabe resolver.
- Los algoritmos vienen dados por su definición escrita y por una serie de imágenes que muestran cada giro.

Aspectos negativos:

- Únicamente se proporcionan los pasos OLL y PLL del método CFOP.
- Los estados de los pasos OLL y PLL solamente muestran un algoritmo cada uno.
- El usuario no puede modificar los algoritmos.

2.4. Conclusiones

Las aplicaciones Android analizadas en esta sección distan, en cuanto a utilidad, de las ya mencionadas applets en Java que permiten al usuario visualizar las rotaciones que componen un algoritmo. Construir una aplicación Android con esta característica supondría introducir en el mercado un producto novedoso que captaría la atención de aquellos que buscan esta solución en sus dispositivos Android.

Por otra parte, la mayoría de las aplicaciones se centran en el método CFOP, e incluso algunas ni siquiera lo abarcan por completo. Otras solo contemplan el método de resolución

para iniciados. Este Trabajo de Fin de Grado se ha diseñado con la suficiente flexibilidad como para permitir al desarrollador incorporar nuevos métodos, pasos y estados haciendo únicamente cambios en el fichero de datos empleado en la carga inicial de la aplicación. Esto lleva a una fácil adopción de las formas de resolución que los usuarios demanden en un futuro. Además, el usuario tendrá libertad para eliminar y añadir algoritmos desde la propia aplicación.

Por otra parte, no solamente basta con proporcionar funcionalidad nueva para hacerse destacar. Ésta debe ir envuelta de una interfaz amigable que enriquezca la experiencia de usuario. Prácticamente todas las aplicaciones analizadas hacen caso omiso de las líneas de diseño estándar proporcionadas por Google. Además, algunas abusan de la publicidad y terminan ofreciendo una experiencia de uso negativa.

3

Definición del proyecto

En esta sección se definen y explican el alcance del proyecto, la metodología de desarrollo escogida y las herramientas utilizadas.

3.1. Alcance

Este Trabajo de Fin de Grado tiene como principal objetivo agilizar el proceso de aprendizaje de los diferentes métodos de resolución del cubo de Rubik. Para ello, se creará una aplicación Android que ofrezca una solución similar a la de las applets analizadas en la sección 2.2: un cubo de Rubik en tres dimensiones que reproduzca automáticamente giros. Asimismo, permitirá que el usuario modifique los algoritmos con total libertad para que se adecúen a su estilo. No obstante, no se dará la posibilidad de modificar los métodos, los pasos ni los estados, teniendo que estar estos incorporados de serie al publicar la aplicación en Google Play.

Por ser una aplicación destinada al uso en dispositivos Android, la interfaz gráfica seguirá las líneas de diseño establecidas por Google para este sistema operativo.

3.2. Metodología

El primer análisis de los requisitos (ver capítulo 4) mostró una serie de características que la aplicación debía incorporar. Además de éstas, muchas otras podían tener cabida. No obstante, es necesario tener en mente que los recursos son limitados y que, tras un tiempo de desarrollo, las ideas iniciales pueden no ser tan válidas o pueden surgir nuevas

que las reemplacen para conseguir un mejor producto. Es por esto que se decidió que el ciclo de vida fuese iterativo incremental. Este modelo consta de diversas etapas en cada iteración:

- Análisis de los requisitos y asignación de prioridades según el estado en el que se encuentre la aplicación.
- Elección de la funcionalidad más importante.
- Diseño gráfico.
- Desarrollo de la funcionalidad escogida para conseguir lo más pronto posible realimentación. Refactorización del código si fuese necesario.
- Pruebas de la funcionalidad implementada y obtención de feedback.
- Reunión con el tutor para —vuelta al primer paso— analizar los requisitos restantes y asignar prioridades y fechas de fin de iteración.

A continuación se detallan las iteraciones que han compuesto el ciclo de vida de la aplicación:

Análisis inicial de requisitos. Las tareas de esta etapa fueron: a) analizar el estado del arte, b) definir los objetivos de la aplicación, c) crear una primera lista de funciones posibles, ordenadas por prioridad. Tras ellas se obtuvo la siguiente salida: lista ordenada por prioridad de toda la posible funcionalidad restante.

Todas las salidas de una iteración fueron la entrada de la siguiente iteración. A partir de ésta no se volverán a mencionar por ser todas las salidas/entradas las que se indican a continuación:

- Aplicación Android actualizada.
- Lista ordenada por prioridad de toda la posible funcionalidad restante.

Además de las etapas comunes a todas las iteraciones, las específicas de cada una fueron las siguientes:

1ª iteración. Construir un cubo de Rubik usando OpenGL ES 2.0.

2ª iteración. Diseñar la base de datos. Desarrollar una estructura interna para poder girar las capas del cubo.

3ª iteración. Desarrollar los métodos necesarios para que el cubo gire automáticamente a partir de una serie de caracteres que definen un algoritmo.

4ª iteración. Desarrollar los métodos necesarios para que el cubo se pinte a partir de una codificación de estado.

5ª iteración. Diseñar e implementar una interfaz gráfica que permite reproducir algoritmos.

6ª iteración. Diseñar e implementar una interfaz gráfica que permite crear nuevos algoritmos.

7ª iteración. Implementar la base de datos, introducir datos en ella y conectarla con las interfaces de creación y reproducción de algoritmos.

8ª iteración. Diseñar e implementar pantallas que contienen métodos de resolución, pasos y estados, así como la navegación entre las mismas.

9ª iteración. Diseñar e implementar una interfaz gráfica que permite manejar copias de seguridad en Dropbox de los algoritmos de la aplicación.

10ª iteración. Diseñar e implementar una interfaz gráfica que permite al usuario girar libremente el cubo de Rubik.

3.3. Herramientas

En esta sección se muestran las diferentes herramientas que se han empleado para la elaboración del presente Trabado de Fin de Grado y que han servido para resolver las siguientes necesidades:

- Programar la aplicación para el sistema operativo Android.
- Probar la aplicación.
- Disponer de un control de versiones.
- Diseñar diagramas y maquetas.
- Realizar copias de seguridad en la nube de los datos de la aplicación.

3.3.1. Entorno para programar aplicaciones Android

El IDE empleado para programar la aplicación ha sido el oficial de Google para programar aplicaciones Android: Android Studio [6]. Está construido sobre IntelliJ IDEA Community Edition [7], un IDE de Java creado por JetBrains. Android Studio es totalmente gratuito.

La aplicación ha sido desarrollada empleando el SDK de Android, cuyo lenguaje de programación es Java. Es posible también programar aplicaciones en código nativo utilizando C o C++, a través del NDK de Android [8]. No obstante, este kit no se ha utilizado, pues su uso es recomendado solamente si la aplicación a desarrollar utiliza de forma intensiva la CPU, tal como hacen los motores de videojuegos o las aplicaciones que realizan simulaciones físicas o procesan señales.

3.3.2. Bibliotecas de código abierto

Además del propio SDK de Android, se han utilizado las siguientes bibliotecas de código abierto liberadas por terceros y disponibles en GitHub:

- `AutoFitTextView`, de Grantland Chew (grantland) [9]. Componente gráfico `TextView` que automáticamente ajusta el tamaño del texto al espacio disponible.
- `CircularProgressBar`, de Antoine Merle (castorflex) [10]. Componente gráfico consistente en un círculo de progreso indeterminado.
- `FButton`, de Le Van Hoang (hoang8f) [11]. Componente gráfico `Button` con estilo *flat*.
- `FloatingActionButton`, de Melnykov Oleksandr (makovkastar) [12]. Componente gráfico FAB (*Floating Action Button*) que se puede asociar a una lista y se oculta al hacer scroll hacia abajo.
- `Material Dialogs`, de Aidan Follestad (afollestad) [13]. Componente gráfico `Dialog` con estilo Material design.
- `Picasso`, de Square [14]. Biblioteca para manejar eficientemente la presentación de gran cantidad de imágenes.
- `SmartTabLayout`, de ogaclejapan [15]. Componente gráfico `ViewPager` con varias opciones de personalización.

3.3.3. Pruebas de la aplicación

La gama de dispositivos Nexus [16] pretende ser una referencia para los desarrolladores a la hora de diseñar sus aplicaciones. Para probar el funcionamiento de la aplicación se ha utilizado un Nexus 5 [17], smartphone desarrollado por Google en colaboración con LG.

Gran parte de los esfuerzos puestos en la aplicación han sido acaparados por el diseño de la misma. En Material design (ver sección 5.3.1), todos los componentes se alinean en una malla de cuadros de 8dp, excepto el texto y ciertos iconos, que lo hacen en una malla de cuadros de 4dp. La aplicación Keyline Pushing [18] dibuja en el dispositivo la malla de 8dp, lo que permite comprobar fácilmente que se cumplen los estándares de diseño.

3.3.4. Control de versiones

El control de versiones permite registrar los cambios realizados sobre una serie de archivos. Es empleado en desarrollos software por varias razones:

- Registrar los cambios realizados para posteriores comprobaciones.

- Minimizar el impacto de un borrado accidental de archivos esenciales.
- Deshacer cambios no deseados en los archivos.

Para el código de la aplicación se ha optado por emplear git como control de versiones y GitHub como plataforma donde alojar los archivos. Esta última ofrece gratuitamente un plan para estudiantes que permite crear proyectos privados [19]. Además, Android Studio integra GitHub, lo que facilita la ejecución de tareas como adición, eliminación y actualización de archivos del repositorio.

3.3.5. Diseño de diagramas y maquetas

El modelo ER de la base de datos se ha creado con el programa Dia [20]. Las maquetas y el resto de diagramas han sido diseñados con Cacoo [21].

3.3.6. Copias de seguridad en la nube de los datos de la aplicación

La aplicación permite al usuario administrar copias de seguridad online de los algoritmos, los cuales son los únicos elementos de un método que el usuario puede modificar en la aplicación. Se ha optado por utilizar la solución proporcionada por Dropbox, pues este servicio online es bastante popular hoy día.

Dropbox ofrece un SDK para Android que abstrae el uso de su API. Actualmente hay tres versiones de este SDK, pero dos de ellas fueron declaradas obsoletas en abril de 2015, por lo que se ha optado por utilizar aquella con soporte: la Core API [22]. Permite conectar la aplicación con la cuenta de Dropbox del usuario para que éste tenga a su disposición sus propias copias de seguridad.

4

Requisitos

En este sección se enumeran los requisitos de la aplicación. Estos se dividen en dos clases: funcionales y no funcionales. Los primeros son declaraciones de los servicios que proveerá la aplicación, mientras que los últimos definen la calidad que se espera de ellos.

Dado el carácter iterativo del ciclo de desarrollo de la aplicación, primero se elaboró una lista de requisitos a partir de la evaluación del estado del arte. Posteriormente, tras cada iteración, dicha lista fue actualizándose incorporando y eliminando requisitos. No obstante, siempre se ha mantenido de tal forma que cumpliese el objetivo principal de la aplicación: cubrir las necesidades no resueltas de aquellos que buscan facilitar su aprendizaje de métodos de resolución del cubo de Rubik.

A continuación se detallan los requisitos.

4.1. Requisitos funcionales

RF. 1 Los métodos de resolución incluidos en la aplicación estarán compuestos de pasos.

RF. 2 Los pasos se mostrarán ordenados.

RF. 3 Los pasos contendrán, cada uno, una serie de posibles estados del cubo de Rubik.

RF. 4 Los estados del cubo de Rubik incluidos dispondrán, como mínimo, de un algoritmo que resuelva el paso al que pertenece cada estado.

RF. 5 El usuario podrá añadir y eliminar algoritmos desde la propia aplicación.

RF. 6 Para cada estado, el usuario podrá elegir su algoritmo preferido.

RF. 7 Los algoritmos se podrán reproducir en un cubo de Rubik tridimensional. El usuario podrá reproducir el algoritmo de tal manera que los giros que lo componen se ejecuten automáticamente uno tras otro, o podrá optar por reproducir únicamente el giro siguiente o anterior, si los hubiere. Además, el usuario podrá pausar la reproducción automática.

RF. 8 El usuario podrá modificar la velocidad de giro del cubo de Rubik.

RF. 9 Para cada estado, el usuario podrá indicar si ha aprendido a resolverlo.

RF. 10 El usuario podrá filtrar los estados según se hayan aprendido o no.

RF. 11 Se proporcionará una pantalla para que el usuario mueva libremente un cubo de Rubik y pueda deshacerlo de manera aleatoria.

RF. 12 Se podrán hacer copias de seguridad de los algoritmos en Dropbox. El usuario podrá conectar la aplicación a su cuenta de Dropbox para subir archivos en formato JSON que contengan los algoritmos que tenga la aplicación en ese momento. Además, el usuario verá una lista de las copias de seguridad que tenga en su cuenta de Dropbox y podrá seleccionar una de ellas para restaurar los algoritmos de la aplicación con los existentes en dicha copia.

RF. 13 El usuario podrá elegir el tema de colores por defecto para el cubo de Rubik tridimensional.

RF. 14 La aplicación se iniciará mostrando el último método de resolución visitado.

4.2. Requisitos no funcionales

RNF. 1 La aplicación será compatible con dispositivos cuyo sistema operativo sea Android 4.1 (Jelly Bean) o superior y cuenten con soporte OpenGL ES 2.0.

RNF. 2 La interfaz gráfica cumplirá los estándares de Material design.

RNF. 3 Se podrá acceder a cualquier estado a través de la elección, primero del método, después del paso que lo contiene.

RNF. 4 Se podrá acceder a cualquier estado desde cualquier parte de la aplicación en menos de 5 segundos.

RNF. 5 La aplicación contendrá los siguientes métodos de resolución: principiante, CFOP y 3OP.

RNF. 6 La interfaz gráfica mostrará sus textos en español si éste es el idioma del dispositivo Android. Lo hará en inglés si el idioma es cualquier otro.

RNF. 7 El movimiento del cubo de Rubik será fluido.

RNF. 8 A la hora de reproducir un algoritmo, se mostrará la cadena de caracteres que lo define. Además, se destacará cuál es el giro que se está reproduciendo en cada momento.

RNF. 9 Todos los pasos y estados irán acompañados de imágenes que permitan discriminarlos.

RNF. 10 Se mostrarán iconos que definan el tipo de cada método de resolución.

5 Diseño

El análisis de los requisitos lleva a diseñar una aplicación Android basada en los siguientes pilares:

- Una base de datos que almacene métodos de resolución, pasos, estados, algoritmos y colores del cubo de Rubik.
- Una interfaz gráfica que organice métodos, pasos y estados y que cuente con un cubo de Rubik tridimensional para la reproducción de algoritmos.

5.1. Patrón de arquitectura

Se ha seguido el patrón de arquitectura modelo-vista-controlador, que separa los datos (modelo) de la interfaz gráfica (vista) y de la lógica que opera con ellos (controlador).

En la figura 5.1 se pueden ver los diferentes módulos que componen el proyecto, cuya implementación se explica en el capítulo 6.

5.2. Base de datos

En la sección 2.1 se definieron los métodos de resolución, pasos, estados y algoritmos. Su existencia determina el diseño de la base de datos, mostrado en la figura 5.2. Además, la base de datos contiene los temas de colores con los que se podrá pintar el cubo de Rubik.

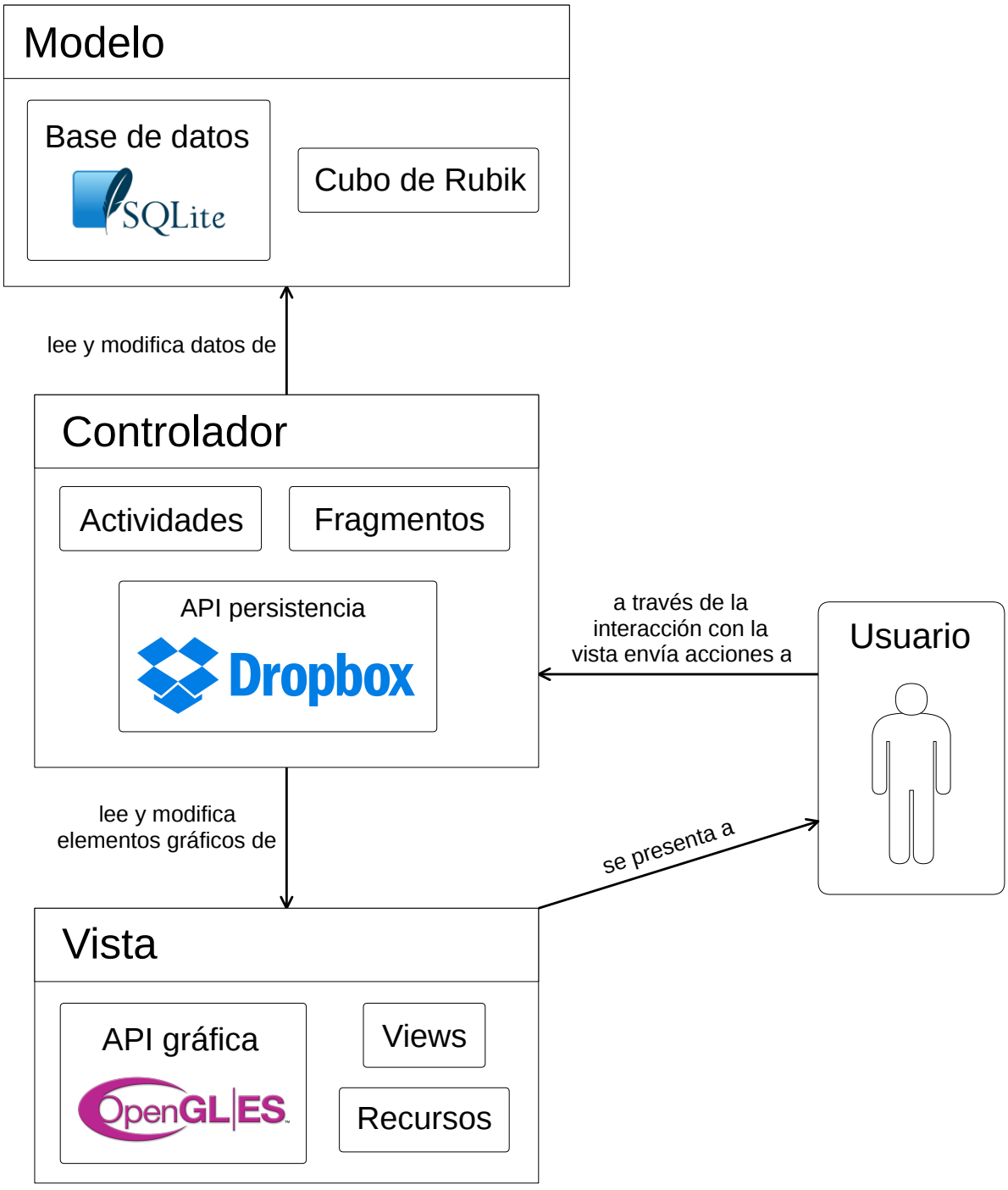

Figura 5.1: Patrón de arquitectura modelo-vista-controlador

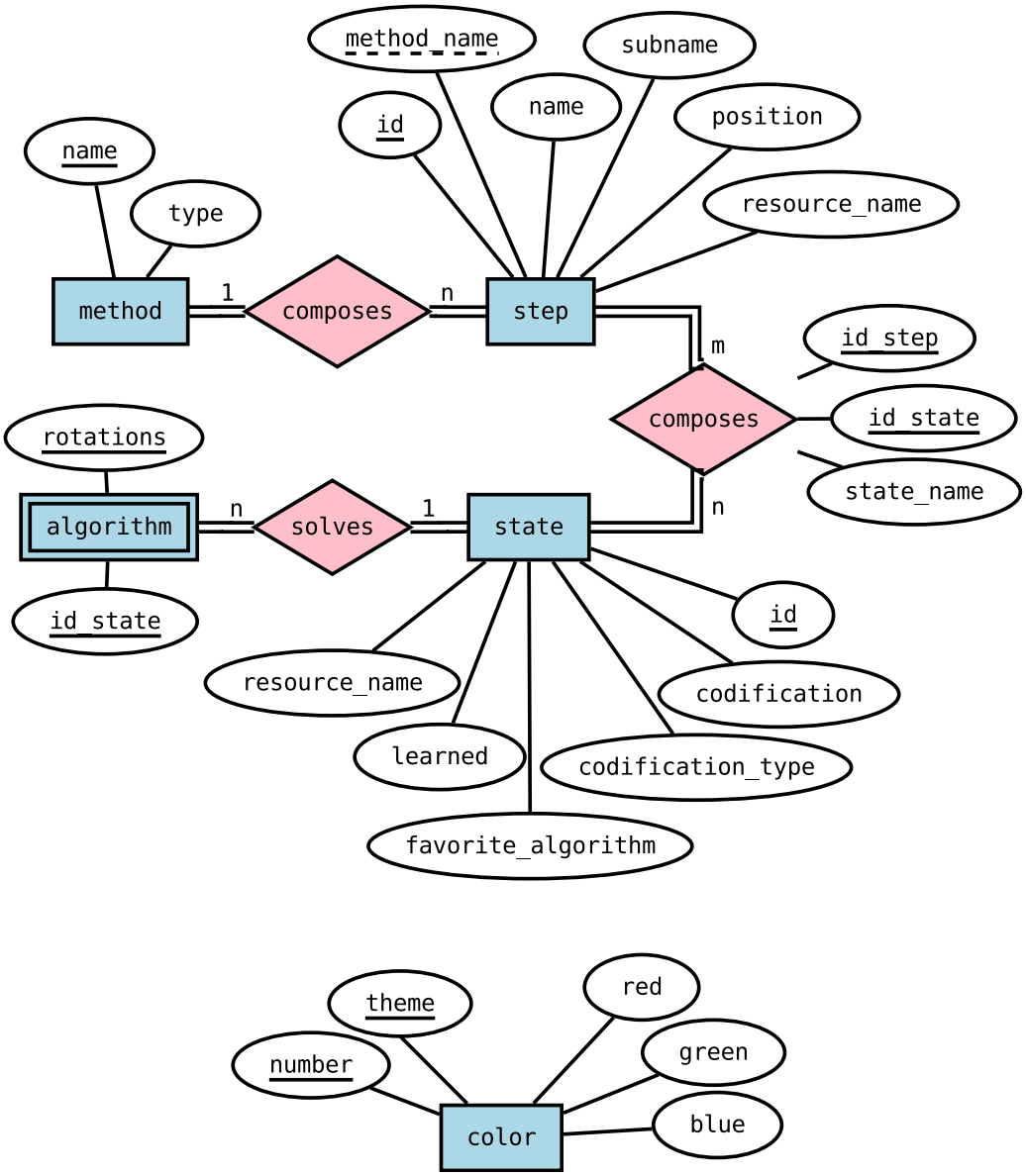

Figura 5.2: Modelo Entidad-Relación de la base de datos

A continuación se enumeran las entidades y sus atributos y se explican las relaciones existentes entre ellas. Ninguno de los atributos puede ser NULL.

Todos los métodos están compuestos por pasos, y todos los pasos componen métodos. De ahí que la participación de ambas entidades en la relación que las vincula sea total. Esta relación es 1-n, pues un método está compuesto por uno o más pasos, mientras que cada paso pertenece únicamente a un método.

- `method`. Representa un método de resolución. Sus atributos son:
  - `name`. Actúa de clave primaria.
  - `type`. Su valor define el tipo de categoría a la que pertenece. Las categorías son: `BEGINNER`, `SPEEDSOLVING` y `BLINDFOLD`.

- `step`. Representa un paso. Sus atributos son:
  - `id`. Identificador numérico como clave primaria.
  - `method_name`. Clave externa que hace referencia a la clave primaria de `method`. Es necesaria por ser la relación 1 `method` - n `step`.
  - `name`. Nombre. Se usa para describir qué resuelve el paso.
  - `subname`. Subnombre. Varios pasos pueden referirse a una misma sección del cubo de Rubik que resuelven parcialmente. El subnombre especifica qué parte resuelve cada uno.
  - `position`. Posición del paso en el método. Es necesario dado que los métodos resuelven secuencialmente el cubo de Rubik.
  - `resource_name`. Cada paso tiene asociado una imagen que lo representa. En este atributo se guarda el nombre de dicha imagen.

Todos los pasos están compuestos por estados, y todos los estados aparecen en pasos. De ahí que la participación de ambas entidades en la relación que las vincula sea total. Esta relación es n-m, pues un paso está compuesto por uno o más estados y un estado puede aparecen en uno o más pasos. Por ello la relación n-m será una tabla que contenga la clave primaria de ambas entidades. Además, cada estado puede aparecer con un nombre distinto según el paso en el que aparezca. Por este motivo, el atributo que representa al nombre del estado se encuentra en esta relación y no en la tabla `state`.

- `steps_states`. Tabla creada a partir de la relación n-m entre pasos y estados.
  - `id_step`. Identificador numérico del paso. Forma parte de la clave primaria.
  - `id_state`. Identificador numérico del estado. Es la otra parte de la clave primaria.
  - `name_state`. Nombre que el estado adquiere en el paso en el que aparece.
- `state`. Representa un estado. Sus atributos son:
  - `id`. Identificador numérico como clave primaria.
  - `codification`. Codificación empleada a la hora de dibujar con OpenGL ES el cubo de Rubik tridimensional. Describe los colores de cada pegatina.
  - `codification_type`. Las codificaciones pueden tener diferente tipo dependiendo de la caracterización de los colores de las pegatinas. Si, por ejemplo, un estado pertenece al paso OLL, obligatoriamente el cubo de Rubik tendrá sus dos primeras capas ya resueltas. Gracias a este atributo, el color de las pegatinas de estas capas ya es conocido, siendo innecesario almacenarlo en la codificación del estado. Sus valores pueden ser: FULL, F2L, OLL o PLL. Ver apéndice C.
  - `favorite_algorithm`. Guarda las rotaciones del algoritmo favorito para resolver este estado. La decisión de que este atributo no pueda ser NULL —al igual que el resto de atributos de la base de datos— viene dada porque un estado debe ser siempre resuelto por un algoritmo. De esta manera, si un estado solamente se soluciona con un algoritmo, éste será el favorito.

- `learned`. Indica si el usuario sabe o no resolver el estado.
- `resource_name`. Cada estado tiene asociado una imagen que lo representa. En este atributo se guarda el nombre de dicha imagen.

Todo estado se resuelve por al menos un algoritmo, y todos los algoritmos resuelven estados. De ahí que la participación de ambas entidades en la relación que las vincula sea total. Si se define un algoritmo únicamente por sus rotaciones, entonces la relación sería n-m, pues un estado se resolvería por uno o más algoritmos y un algoritmo podría resolver uno o más estados. Como la relación n-m implicaría la creación de una tabla que tuviese la clave primaria de `algorithm` (`rotations`) y la de `state` (`id`), y como `algorithm` solamente contiene las rotaciones que lo componen, entonces esta relación puede ser transformada en una relación n `algorithm` - 1 `state` en la que la tabla `algorithm` contenga la clave primaria del estado que resuelve.

- `algorithm`. Representa un algoritmo. Sus atributos son:
  - `rotations`. Serie de rotaciones que componen el algoritmo. Forma parte de la clave primaria.
  - `id_state`. Identificador del estado que resuelve. Es la otra parte de la clave primaria.

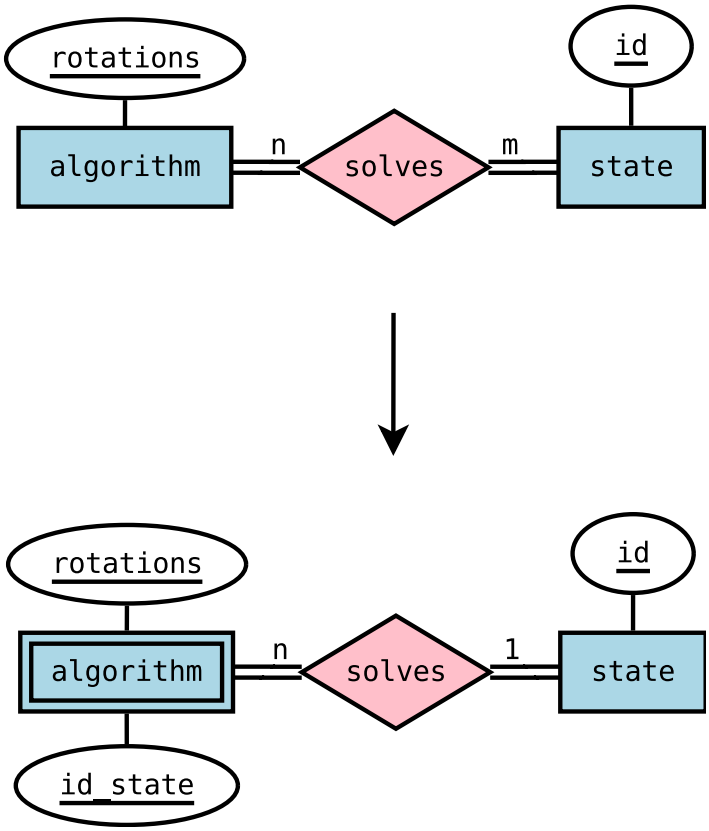

Figura 5.3: Paso de una relación n `algorithm` - m `state` a una relación n - 1

- `color`. Representa el color de una cara del cubo de Rubik. Sus atributos son:
  - `number`. Identificador numérico de la cara del cubo de Rubik.
  - `theme`. Dado que no todas las personas están acostumbradas a resolver el cubo de Rubik comenzando por una cara determinada, se da la posibilidad de que se escoja el tema de colores preferido. Por ejemplo, el color rojo en el tema 1 se identifica con la cara 3, mientras que en el tema 2 lo hace con la cara 4.
  - `red`. Intensidad del color rojo en la notación RGB. Su valor pertenece al rango [0, 1].
  - `green`. Intensidad del color verde en la notación RGB. Su valor pertenece al rango [0, 1].
  - `blue`. Intensidad del color azul en la notación RGB. Su valor pertenece al rango [0, 1].

5.3. Interfaz gráfica

Google ha dirigido gran parte de sus esfuerzos relativos a Android a que las aplicaciones de este sistema operativo sigan unas líneas de diseño gráfico estándar. Para conseguir que los desarrolladores tengan una guía de referencia para crear o adaptar sus aplicaciones, Google lanzó Material design en su conferencia anual Google I/0, en junio de 2014 [23].

Material design se caracteriza, entre otras cosas, por:

- Crear sensación de profundidad mediante el uso de sombras. De esta manera es posible destacar unos elementos sobre otros y agrupar componentes según niveles de profundidad. Los usuarios entienden que un elemento que reciba sombra de otro estará por debajo.
- Dar importancia al estilo. Esto incluye pautas como que los iconos tienen un estilo plano o que se deben usar dos colores que se diferencien bien: uno como principal y otro para acentuar.
- Las transiciones entre pantallas son lógicas. El usuario sabe así por qué la acción que realiza tiene un determinado resultado.
- La diversidad de componentes gráficos, cada uno de los cuales se usa según un determinado propósito.

5.3.1. Material design en la aplicación

La aplicación se ha diseñado, en la medida de lo posible, según los patrones definidos por Material design [24]. A continuación se detallan los componentes empleados en el diseño:

- Bottom sheet. Lámina oculta que se muestra deslizándose desde la parte inferior de la pantalla.
- Botones. Muestran qué acción ocurrirá al ser pulsados. Su color debe ser el de acentuación en caso de querer destacarlo.
- FAB (*Floating Action Button*). Es un tipo de botón especial usado para destacar una acción. Se distingue del resto de botones por ser circular y tener cierta elevación. Su color debe ser el de acentuación.
- Cards. Contienen información variada, no tienen que seguir el mismo diseño entre ellas y sirven como punto de entrada a información más detallada.
- Círculo de progreso indeterminado. Indica que se está realizando una acción y que el usuario debe esperar a que ésta termine.
- Dialogs. Muestran información sobre una acción o un hecho crítico y requieren que el usuario tome una decisión.
- Listas. Presentan múltiples elementos ordenados verticalmente. Los elementos deben tener el mismo diseño.
- Navigation drawer. Panel lateral que se encuentra oculto y que el usuario puede mostrar mediante el gesto de deslizar el dedo desde el borde izquierdo de la pantalla o pulsando un icono situado en la toolbar.
- Pestañas. Organizan la información en diferentes categorías.
- Snackbars. Muestran al usuario un pequeño texto a modo de feedback de una operación que acabe de realizar. Son barras que aparecen en la parte inferior de la pantalla brevemente.
- Switches. Sirven para alternar entre dos estados.
- Toolbar. Barra situada en la parte superior de la pantalla y por encima del cuerpo de la misma. Su color es el principal de la aplicación. Como la aplicación cuenta con un navigation drawer, la toolbar contiene un icono para indicar su existencia y cuya acción al ser pulsado es mostrarlo.

5.3.2. Maquetas

La figura 5.4 muestra las maquetas de la interfaz gráfica de la aplicación. En ellas se aprecia las diferentes pantallas de la que constará la aplicación, así como la navegación entre las mismas.

A continuación se explica la motivación de cada pantalla:

A. Dada la estructura métodos/pasos/estados, la aplicación debe proporcionar una navegación entre estos elementos. Los métodos, como son pocos, se colocan en el navigation drawer.

B. Al seleccionar un método, la pantalla muestra una lista de los pasos que lo componen. Cada paso se representa por una imagen, un nombre y un subnombre.

C. Tras elegir un paso, el usuario verá los estados que pertenecen a él. Cada estado se representa por una imagen y un nombre; además, se indica si el usuario lo ha aprendido. Esta pantalla incorpora tres pestañas para filtrar o no los estados según se hayan aprendido o no.

D. Cuando el usuario elige un estado, se muestra la pantalla que representa el objetivo principal de la aplicación, el cual es visualizar la ejecución de algoritmos en un cubo de Rubik tridimensional. Para ello, la pantalla posee dicho cubo y un reproductor de algoritmos. Se ha diseñado el reproductor para que dé la posibilidad de visualizar de manera secuencial la ejecución de las rotaciones, ya sea de manera automática reproduciéndose cada giro al finalizar el anterior, o de manera manual eligiendo ejecutar el siguiente o el anterior giro.

E. Dado que un estado se resuelve con uno o más algoritmos, se debe crear una lista para que el usuario seleccione su algoritmo preferido para resolver un determinado estado. Este algoritmo será el que se muestre en el reproductor.

F. El usuario debe poder crear algoritmos. La pantalla de creación de los mismos cuenta con una serie de botones, cada uno asociado a un determinado giro. Mediante estos botones, el usuario puede ir construyendo su algoritmo para posteriormente guardarlo en la base de datos.

G. Para que el usuario mueva libremente un cubo de Rubik y pueda deshacerlo de manera aleatoria, existirá una pantalla que muestre el cubo tridimensional y botones para poder girarlo libremente, además de uno que lo deshaga aleatoriamente. A esta pantalla se accede desde el navigation drawer.

H. Otro de los requisitos de la aplicación es permitir realizar copias de seguridad en la cuenta de Dropbox del usuario. La gestión de las mismas se llevará a cabo en una pantalla independiente, a la que se puede acceder a través del navigation drawer. En ella, el usuario puede realizar copias de seguridad, ver cuáles tiene en su cuenta de Dropbox y seleccionar una de ella para poder descargarla y volcar su contenido en la aplicación.

I. Por último, el usuario podrá, también a través del navigation drawer, acceder a una pantalla de ajustes en la que podrá elegir el tema de colores del cubo de Rubik. Además, en ella se mostrará información sobre el autor y las bibliotecas de código libre empleadas. A esta pantalla se accede desde el navigation drawer.

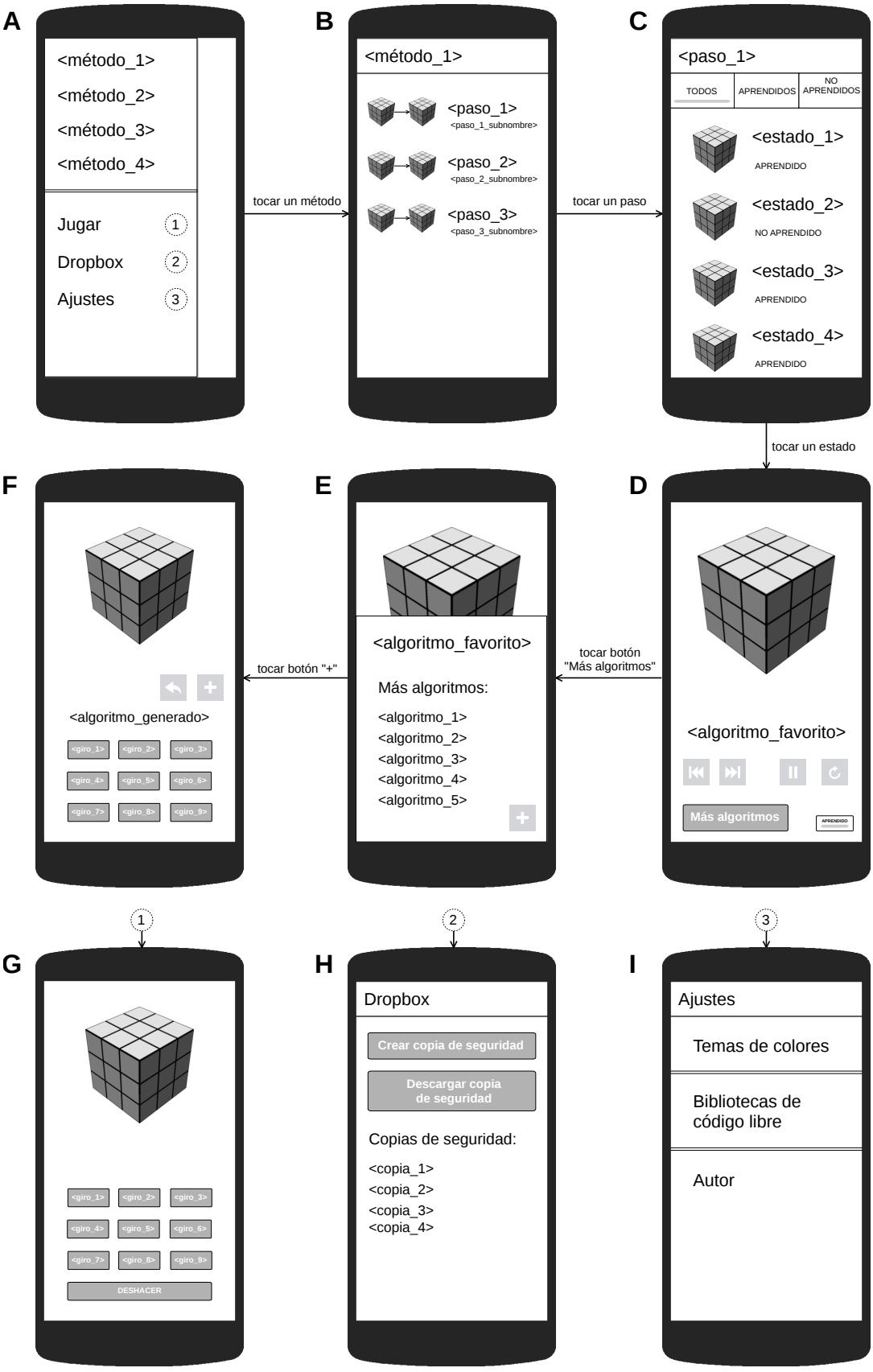

Figura 5.4: Maquetas de la interfaz gráfica de la aplicación

6

Implementación

En esta sección se muestra la estructura del proyecto y se explica la implementación de los diferentes módulos que conforman el patrón Modelo-Vista-Controlador expuesto en la sección 5.1, cuyo esquema es:

- Modelo:
  - Base de datos.
  - Cubo de Rubik.
- Vista:
  - Recursos.
  - Views.
  - OpenGL ES.
- Controlador:
  - Actividades y fragmentos.
  - API de Dropbox.

Gran parte de la información necesaria para la implementación de la aplicación se ha extraído de las guías oficiales de Android [25].

6.1. Estructura del proyecto

El proyecto Android creado a través de Android Studio contiene todo el código fuente y demás archivos necesarios para la construcción de la aplicación. La estructura de la carpeta `src/`, que contiene el código fuente, es la siguiente:

- `main/`. Incluye las carpetas `assets/`, `java/` y `res/`, además del archivo `AndroidManifest.xml`, que describe las características fundamentales de la aplicación y define sus pantallas.
  - `assets/`. Contiene únicamente el archivo `dropbox.properties`, en el cual se guardan los credenciales de la aplicación proporcionados por Dropbox.
  - `java/`. Incluye el paquete principal, dividido en:
    - `controller/`. Formado por las clases `Activity` y `Fragment`. Ver subsección 6.4.1.
    - `model/`. Dividido en:
      - `database/`. Clases relativas al cubo de Rubik. Ver subsección 6.2.1.
      - `rubikscube/`. Clases relativas a la base de datos. Ver subsección 6.2.2.
    - `util/`. Clases de tipo `util`, que aportan una funcionalidad genérica.
    - `view`. Dividido en:
      - `opengl/`. Clases relativas a la construcción del cubo de Rubik con OpenGL ES. Ver subsección 6.3.3.
      - `views/`. Vistas creadas para la aplicación. Ver subsección 6.3.2.
  - `res/`. Carpeta en la que se almacenan los recursos de la aplicación, explicados en la subsección 6.3.1.
- `test/`. Contiene los test unitarios de caja negra de las clases de tipo `util`. Ver sección 7.2.

6.2. Modelo

El modelo de la aplicación gestiona el almacenamiento, consulta y actualización de la información. La mayor parte de ella se encuentra en base de datos; el resto se ha definido como enumerados.

6.2.1. Base de datos

La base de datos, detallada en la sección 5.2, se ha implementado en lenguaje SQLite. La elección de este tipo de base de datos viene motivada por ser SQLite el que las guías de desarrollo en Android recomiendan. El paquete `android.database.sqlite` incorpora todas las herramientas necesarias para la creación y gestión de bases de datos SQLite;

en particular, se ha extendido la clase `SQLiteOpenHelper` para ello. En esta clase hay métodos públicos que son llamados por el controlador para acceder a los datos y realizar las actualizaciones oportunas.

Cada vez que se abre la base de datos es necesario cerrarla después de su uso. Si no se hiciera y se intentase volver a abrirla, se recibiría la siguiente excepción: «java.lang.IllegalStateException: SQLiteDatabase created and never closed». Para evitar abrir y cerrarla constantemente, se ha hecho uso del patrón Singleton. Con este patrón se consigue tener una única instancia de la clase. Así, la primera vez que se llama a la base de datos, ésta internamente se abrirá. Sucesivas llamadas devolverán la base de datos ya abierta.

Cuando la base de datos se crea, se llena a partir de la información contenida en un archivo con formato JSON, cuya estructura es:

```
{
  "methods":
  [
    {
      "name": <string>,
      "type": <string>
    },
    [...]
  ],
  "steps":
  [
    {
      "id": <number>,
      "method": <string>,
      "name": <string>
      "subname": <string>,
      "pos": <number>,
      "res": <string>
    },
    [...]
  ],
  "states":
  [
    {
      "cod": <string>,
      "cod_type": <string>,
      "res": <string>,
      "fav_alg": <string>,
      "algs": [<string>, [...]],
```

```
      "steps":
      [
        {
          "id": <number>,
          "state_name": <string>
        },
        [...]
      ]
    },
    [...]
  ],
  "colors":
  [
    {
      "theme": <number>,
      "number": <number>,
      "html": <html_string>
    },
    [...]
  ]
}
```

Se ha optado por incorporar los datos desde este archivo principalmente para separarlos del código de la base de datos. Además, la organización de los mismos en la estructura de un archivo JSON facilita la modificación del contenido.

En algunos casos, el nombre de los atributos del archivo JSON es el mismo que el nombre de la columna de la base de datos correspondiente. En otros, cuando este último nombre es demasiado largo, se usan abreviaturas para acortar la longitud del archivo, de tal manera que la correspondencia es la siguiente:

- "`steps`"/"`method`" es "`method_name`".
- "`steps`"/"`pos`" es "`position`".
- "`steps`"/"`res`" es "`resource_name`".
- "`states`"/"`cod`" es "`codification`".
- "`states`"/"`cod_type`" es "`codification_type`".
- "`states`"/"`res`" es "`resource_name`".
- "`states`"/"`fav_alg`" es "`favorite_algorithm`".

Se puede apreciar que el identificador del estado no viene dado en el archivo JSON. Al indicar en cada objeto `states` los algoritmos que lo resuelven y los pasos en los que aparece

y con qué nombre en cada uno, no es necesario tener en este archivo un identificador que los relacione. A la hora de leer el archivo para crear la base de datos, este identificador se crea dinámicamente y se asigna a las tablas `state`, `algorithm` y `steps_states`".

Por otra parte, la tabla `color`, tiene tres columnas: `red`, `green` y `blue` que, como se dijo en la sección 5.2, almacenan la intensidad de cada color en la notación RGB. Sin embargo, en el archivo JSON se especifica el color en HTML. Esto es así porque este código de colores es el más extendido.

En la base de datos los colores se almacenan en formato RGB porque estos valores son los aceptados por OpenGL ES a la hora de dar color a las superficies. En lugar de almacenar un color HTML y tener que convertirlo a RGB cada vez que se quiere pintar, es más eficiente hacer el cálculo una única vez —al leer el JSON y crear la base de datos—.

6.2.2. Cubo de Rubik

Se vio en la sección 5.2 que las entidades relacionadas con la resolución del cubo de Rubik (es decir, los métodos de resolución, los pasos, los estados y los algoritmos) están guardadas en la base de datos. Esto es así porque su cantidad no es estática y lo natural es que varíen con frecuencia, en especial los algoritmos, que pueden ser eliminados y creados por el usuario desde la propia aplicación.

Hay otro tipo de datos que no están guardados en la base de datos porque simplemente consisten en un conjunto de valores predefinidos no variable. Estos tipos se han definido como enumerados y son los siguientes:

- `MethodType`. La entidad `method` tiene un atributo `type`. Como el tipo de un método puede ser únicamente uno de los siguientes: `BEGINNER`, `SPEEDSOLVING` y `BLINDFOLD`, estos valores se guardan en un `enum` llamado `MethodType`. En la aplicación, los métodos se encuentran en el navigation drawer; al lado de cada nombre de método se muestra un icono que representa su tipo. Ver apéndice A.

- `CodificationType`. Son los valores que puede tomar el atributo `codification_type` de la entidad `state`: `FULL`, `F2L`, `OLL` o `PLL`. Ver apéndice C.

- `RotationType`. Representa los diferentes giros de un cubo de Rubik. Ver apéndice B.

6.3. Vista

La vista de la aplicación es la interfaz gráfica que se presenta al usuario. Hace uso de recursos, views y OpenGL ES.

6.3.1. Recursos

En Android es posible definir la interfaz gráfica en una serie de archivos llamados «recursos», de tal manera que ésta sea totalmente independiente de la lógica de la aplicación.

Hay varios tipos de recursos organizados en diferentes carpetas que a continuación se analizan. La mayoría de los recursos son archivos XML en los que cada elemento tiene un atributo que lo identifica inequívocamente y que sirve para poder hacer referencia al recurso externamente.

Se enumeran a continuación las diferentes carpetas de recursos empleadas en la aplicación:

- `color/`. Contiene los XML con los colores empleados en la aplicación.
- `drawable/` y derivados. Incluye todas las imágenes de la aplicación: las de los pasos, los estados y los controles de reproducción, además de los iconos empleados en el navigation drawer. Hay diferentes carpetas de este tipo:
  - Ordenadas de menor a mayor resolución: `drawable-mdpi`, `drawable-hdpi`, `drawable-xdpi`, `drawable-xxdpi` y `drawable-xxxdpi`. En ellas se encuentran las imágenes cuya resolución está adaptada a cada tipo de pantalla según su densidad. *dpi* quiere decir «dots per inch»; es una medida de la cantidad de píxeles en un área de pantalla. Las imágenes de `drawable-xxxdpi` son las que más resolución tienen para así poder verse correctamente en pantallas con una densidad de píxeles altísima; justo al contrario que las imágenes de `drawable-mdpi`.
  - `drawable-nodpi`. Contiene las imágenes que no van a ser reescaladas por el sistema operativo.
- `layout/`. Aquí se encuentran todos los archivos XML que definen las diferentes pantallas de la aplicación. A pesar de que la interfaz gráfica puede generarse mediante código Java, todas las pantallas están definidas en estos archivos. Esto otorga independencia entre módulos y facilita increíblemente la creación, edición y adaptación de la interfaz a los diferentes dispositivos, pues Android Studio incorpora un editor que permite visualizar cada layout en una gran cantidad de terminales de diferentes tamaños y resoluciones. No obstante, elementos como el cubo de Rubik tridimensional deben generarse dinámicamente a través de código Java. Otros, como las listas de elementos, pueden definirse a través de XML pero deben ser rellenados por el controlador, encargado de recoger los datos del modelo.
- `raw/`. Contiene los programas empleados por OpenGL ES y el archivo JSON con los datos a incorporar en la base de datos.
- `values/`. Mientras que los archivos de otros directorios definen un único recurso (por ejemplo, un layout define una pantalla), los XML de esta carpeta describen múltiples recursos. Por cada archivo de este directorio, cada elemento hijo define un

recurso. Por ejemplo, un elemento `string` crea un recurso `R.string` y un elemento `color` crea un recurso `R.color`. Los XML existentes son:

- `arrays.xml`. Contiene la lista empleada en la pantalla de ajustes para mostrar los diferentes temas de colores del cubo de Rubik posibles.
- `colors.xml`. Contiene los colores.
- `dimens.xml`. Contiene las medidas de los componentes gráficos.
- `strings.xml`. Contiene el texto en inglés que aparece en la interfaz gráfica.
- `strings-es.xml`. Contiene el texto en español que aparece en la interfaz gráfica. Si el idioma del dispositivo donde se instale la aplicación es español, se emplearán estas cadenas de caracteres. En caso contrario, se mostrarán las de `strings.xml`.
- `styles.xml`. Contiene los estilos que se aplican de manera específica a algunos componentes gráficos.
- `themes.xml`. Contiene las propiedades de estilo de la aplicación a nivel general. Es en este archivo donde se define el color principal y el de acentuación, así como el color por defecto de los fondos de las pantallas. De esta manera, no es necesario indicar estos colores componente a componente.

- `xml/`. Incluye el archivo `preferences.xml`, un XML que da forma a la pantalla de ajustes.

6.3.2. Views

Las views son componentes gráficos utilizados principalmente en los archivos XML para construir la interfaz gráfica. El SDK de Android proporciona una gran cantidad de ellos; no obstante, a veces resulta práctico extender alguno de estos componentes para poder añadir nueva funcionalidad.

Las views creadas para la aplicación son:

- `MoreAlgorithmsRelativeLayout`. Extiende `RelativeLayout` —uno de los layouts por defecto de Android— para permitir ser mostrada y ocultada, a través de una transición, cuando el usuario hace el gesto de deslizar hacia abajo. Ver figura 6.1.
- `MyFButton`. Extiende `FButton` (ver subsección 3.3.2) para que al activarse o desactivarse, el color y la sombra del botón cambien automáticamente. Ver figura 6.2.
- `SquareImageView`. Extiende `ImageView` para conseguir imágenes cuadradas. Ver figura 6.3.
- `SquareButton`. Extiende `Button` para conseguir botones cuadrados. Ver figura 6.4.

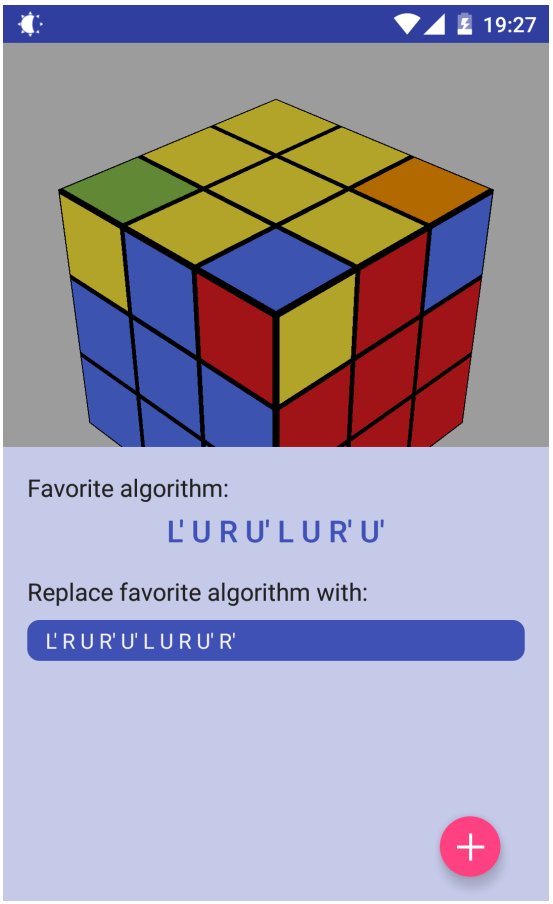

Figura 6.1: Pantalla que contiene una vista `MoreAlgorithmsRelativeLayout`

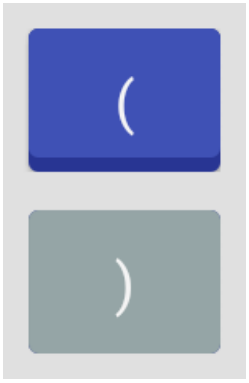

Figura 6.2: Dos vistas `MyFButton`. La superior está activada y la inferior desactivada.

Figura 6.3: Card que contiene una vista `SquareImageView`

Figura 6.4: Vista `SquareButton`

6.3.3. OpenGL ES 2.0

OpenGL es una especificación estándar que define una API para desarrollar aplicaciones con gráficos en dos y en tres dimensiones. OpenGL ES es una variante simplificada de OpenGL, diseñada para dispositivos tales como smartphones o consolas de videojuegos. Android da soporte a diferentes versiones de la API de OpenGL ES:

- OpenGL ES 1.0 y 1.1. Desde la versión 1.0 de Android.
- OpenGL ES 2.0. Desde la versión 2.2 de Android.
- OpenGL ES 3.0. Desde la versión 4.3 de Android.
- OpenGL ES 3.1. Desde la versión 5.0 de Android.

Dado que la aplicación se ha diseñado para poder ejecutarse en dispositivos con Android 4.1 o superior, no se han podido emplear las versiones 3.0 ni 3.1. Se ha hecho uso de la versión 2.0 de OpenGL ES por ser ésta la más reciente de las disponibles.

Prácticamente todo el conocimiento sobre OpenGL ES ha sido adquirido del libro *OpenGL ES 2.0 for Android* [26], de Kevin Brothaler. A continuación se explican algunos principios básicos de OpenGL ES 2.0, fundamentales para entender cómo se ha conseguido construir un cubo de Rubik tridimensional:

colores. El modelo de color de OpenGL es RGB (Red, Green, Blue), según el cual el color está compuesto en términos de la intensidad de los colores primarios de la luz: rojo, verde y azul. Estos colores toman valores en el rango [0, 1], donde 0 representa la ausencia de ese color y 1 representa la máxima intensidad. OpenGL asume que estos colores tienen una relación lineal entre ellos: un valor rojo de 0.5 tendrá dos veces el brillo que un valor rojo de 0.25, pero la mitad que un valor rojo de 1.

vértices. Un vértice es un punto que representa la esquina de un objeto geométrico. Tiene dos atributos:

- Posición. Coordenadas x, y, z del espacio tridimensional. Toman valores en el rango [-1, 1].
- Color. Intensidades del rojo, verde y azul. Toman valores en el rango [0, 1].

La posición y el color hacen que cada vértice esté compuesto por seis `floats`: tres para la posición y tres para el color.

puntos, líneas y triángulos. En OpenGL solamente se pueden dibujar puntos, líneas y triángulos. Todas las superficies son, por tanto, combinaciones de triángulos. Para dibujar, por ejemplo, la pegatina de un cubito, es necesario dibujar dos triángulos que, al compartir la diagonal, formen un cuadrado.

canal (*pipeline*). Los vértices definidos para pintar objetos geométricos pasan a través de un canal que los procesa a través de *shaders* para, finalmente, mostrarlos en pantalla.

shaders. La principal diferencia entre OpenGL ES 1.0/1.1 y OpenGL ES 2.0 es la existencia de unos programas gráficos llamados «shaders». Antes de que apareciesen, OpenGL proporcionaba un conjunto fijo de funciones con los que controlar un número limitado de parámetros, como por ejemplo las luces. Esta API fija era fácil de utilizar, pero difícil de extender si se requería funcionalidad extra como poder crear efectos personalizados. Con la llegada de OpenGL ES 2.0 se añadió una API programable a través del uso de *shaders* y se eliminó completamente la API fija, por lo que el uso de *shaders* es obligatorio en esta versión. Con los *shaders* es posible controlar cómo se pinta cada vértice. De esta manera, se puede crear cualquier efecto personalizado que se quiera, siempre y cuando lo permita GLSL (OpenGL Shading Language), el lenguaje para programar *shaders*. No obstante, para dibujar el cubo de Rubik solamente se han empleado los *shaders* para generar las posiciones de los vértices y para pintar superficies de un único color cada una. Hay dos tipos de *shaders*: *vertex shader* y *fragment shader*.

vertex shader. Se ejecuta una vez por cada vértice y genera su posición final. Una vez que las posiciones de todos los vértices son conocidas, OpenGL cogerá el conjunto de vértices visibles y los transformará en puntos, líneas o triángulos.

fragment shader. Se ejecuta una vez por cada fragmento de cada punto, línea y triángulo y genera su color final. Un fragmento es una pequeña área rectangular, análoga a un píxel de una pantalla. Si, por ejemplo, los dos vértices de una línea tienen el mismo color, entonces todos los fragmentos de la línea serán de ese color. Sin embargo, si los dos vértices tienen colores diferentes, la línea no tendrá un color uniforme, sino un degradado entre ambos colores (ver figura 6.5, en la que se muestra este hecho en un triángulo). El color de los fragmentos no será, por tanto, el mismo, y habrá que calcular qué color corresponde a cada uno según la distancia a cada vértice. Como el color depende de dicha distancia, es necesario que en el canal de OpenGL se ejecute primero el *vertex shader* y después el *fragment shader*.

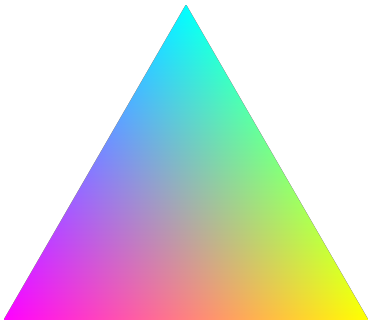

Figura 6.5: Triángulo en el que cada vértice tiene un color distinto. El color de los fragmentos que componen el triángulo viene dado por la distancia a cada vértice. Imagen tomada del libro *OpenGL ES 2.0 for Android* [26], de Kevin Brothaler.

programa OpenGL. Un *programa OpenGL* es simplemente un *vertex shader* y un *fragment shader* enlazados en un único objeto. Los *vertex shaders* y los *fragment shaders* van siempre juntos; ambos son necesarios para generar la imagen final en la pantalla. Sin embargo, aunque vayan juntos, un *shader* cualquiera puede enlazarse

con otros *shaders* en otros programas al mismo tiempo. El proceso de creación del *programa OpenGL* es el siguiente:

a) Se crean dos cadenas de caracteres (`String`). Una de ellas contendrá el código del programa *vertex shader*; la otra, el del programa *fragment shader*.

b) Se crean dos *shader objects*. A uno de ellos se le incluye el `String` con el programa *vertex shader*; al otro, el `String` con el programa *fragment shader*.

c) Se compilan los *shader objects*. Se obtiene como resultado un programa *vertex shader* y un programa *fragment shader*.

d) Se crea un *program object* al cual se le asignan los programas *vertex shader* y *fragment shader* ya compilados.

e) Se enlazan los *shaders* en el programa. Se obtiene así el *programa OpenGL* deseado.

Una vez que el *programa OpenGL* se ha creado, para pintar objetos geométricos a partir de vértices hay que seguir los siguientes pasos:

1. Indicar qué *programa OpenGL* se va a usar.
2. Definir un array con las tuplas de seis floats de cada vértice.
3. Asignar el array al *programa OpenGL*.
4. Definir cómo van emparejados los vértices, si en puntos, líneas o triángulos.
5. Dibujar la vista OpenGL a partir del contenido del *programa OpenGL*.

Un cubo de Rubik tiene 26 cubitos, de los cuales 6 son centros. Cada centro consta de 2 cuadrados: la propia cara del centro y la pegatina. El resto de cubitos constan de 6 caras cada uno. Si el cubito es una esquina tendrá, además, 3 pegatinas; si es una arista, tendrá 2. Tanto las caras como las pegatinas son cuadradas, por lo que es necesario dibujar 2 triángulos para formar cada una. Para construir un cubo de Rubik hay que definir, por tanto, de manera exacta las coordenadas de los vértices de todos estos triángulos. Además, hay que asignarles un color determinado, pues todas las caras estarán pintadas de negro, pero las pegatinas tendrán cada una el color que indique la codificación del cubo de Rubik a pintar.

No solamente es necesario pintar un cubo de Rubik, también hay que ser capaz de girar sus diferentes capas. Para ello es necesario usar una matriz de rotación. Las coordenadas de un vértice en el espacio tridimensional forman un vector que, al multiplicarse por una matriz de rotación, se transforma en otro vector con nuevas coordenadas. Por otra parte, los giros se pueden realizar a menor o mayor velocidad. Cuando se gira 90º una capa a una velocidad lenta, lo que se está haciendo en realidad es dibujar 90 veces el cubo de Rubik. En cada iteración la matriz de rotación tendrá valores diferentes que se irán modificando de tal manera que, al multiplicar al vector de coordenadas del vértice, éste experimente una rotación de, cada vez, un grado más. En la última iteración, la rotación será de 90º

y se completará el giro. Si se quiere rotar a una velocidad rápida, basta indicar que el incremento sea de 6 grados; así, solamente se sucederán 15 dibujos del cubo de Rubik y dará la impresión de que la capa se está girando a mayor velocidad.

6.4. Controlador

El controlador recibe las acciones del usuario producidas al interactuar con la vista. Acude al modelo para leer o modificar datos y actualiza la vista.

6.4.1. Actividades y fragmentos

Una actividad es una clase `Activity` que provee una ventana. Las aplicaciones Android normalmente están compuestas de diferentes actividades, en cada una de las cuales se dibuja una interfaz gráfica.

El patrón de arquitectura Modelo-Vista-Controlador en Android es tratado realmente como Modelo-Vista-Actividad, pues las actividades actúan de controlador por ser las encargadas de pintar la vista, acudir al modelo para obtener y modificar datos y por recoger las acciones del usuario y actuar en consecuencia, ya sea actualizando la vista o el modelo.

Las actividades se relacionan entre sí en tanto en cuanto cada una puede iniciar otra y así permitir una navegación entre diferentes pantallas. Cuando una nueva actividad se inicia, la anterior se para, pero el sistema no la elimina, sino que la mantiene en una pila, la «back stack», de tal manera que cuando el usuario presiona el botón «atrás», el comportamiento por defecto es destruir la actual actividad y recuperar la anterior de la pila.

Cuando una actividad se para debido a que otra nueva comienza, el objeto `Activity` a parar es notificado de este cambio en su estado a través de un método determinado. Las actividades tienen un ciclo de vida y el cambio entre sus estados es anunciado a través de este tipo de métodos.

Además de las actividades, los fragmentos (clase `Fragment`) actúan también de controlador. Se caracterizan porque una actividad puede incluir diferentes fragmentos. El objetivo de estos es, por tanto, ser reusados en diferentes actividades y evitar así la duplicación de código.

Al igual que las actividades, los fragmentos tienen su propio ciclo de vida.

El ciclo de vida de un fragmento está afectado directamente por el ciclo de vida de la actividad que lo contiene. Si una actividad es pausada, sus fragmentos también lo son; si se para, sus fragmentos se paran.

La aplicación se ha construido aprovechando las ventajas que ofrecen los fragmentos. Además de las comentadas, otra ventaja es que una actividad puede reemplazar un

fragmento por otro mediante una transacción. Al hacer esta transacción, es posible mandar el fragmento reemplaza a una pila, de tal manera que cuando el usuario presiona el botón «atrás», se puede recuperar de la pila y deshacer la transacción.

En la aplicación se han usado tres actividades y ocho fragmentos distintos, como se ve en la figura 6.6.

Los fragmentos empleados son:

- `StepsFragment`. Su constructor recibe el nombre de un método y muestra una lista de los pasos que lo componen.
- `StatesFragment`. Su constructor recibe el identificador de un paso y muestra una lista de los posibles estados que pueden aparecer en dicho paso. La pantalla tiene tres pestañas que permiten filtrar los estados según el usuario los haya aprendido o no.
- `MyPreferenceFragment`. Extiende la clase `PreferenceFragment`, que es un fragmento que dibuja su interfaz a partir de un archivo XML con una estructura de preferencias.
- `DropboxFragment`. Permite conectar la aplicación a la cuenta de Dropbox del usuario para realizar copias de seguridad de los algoritmos.
- `RubiksCubeFragment`. Su constructor recibe el identificador de un estado y se encarga de crear una vista en la que dibujar un cubo de Rubik tridimensional que modelice dicho estado.
- `AlgorithmPlayerFragment`. Muestra el conjunto de rotaciones que forman un algoritmo y botones para manejar su reproducción.
- `AlgorithmCreatorFragment`. Dibuja un layout que muestra 36 botones, cada uno asociado a uno de los posibles giros del cubo de Rubik (ver apéndice B). Su propósito es recoger la pulsación de cada botón para ir mostrando el algoritmo que el usuario va generando y permitir guardarlo en la base de datos.
- `PlayRotationsFragment`. Al igual que `AlgorithmCreatorFragment`, dibuja un layout que muestra 36 botones, cada uno asociado a uno de los posibles giros del cubo de Rubik. Sin embargo, no se emplea para crear algoritmos, por lo que no se va escribiendo ninguno, sino que sirve para que el usuario ejecute rotaciones a placer. Tiene, además, un botón para deshacer el cubo mediante giros aleatorios.

Las actividades que componen la aplicación son:

- `MainActivity`. Es la actividad que se muestra nada más ejecutar la aplicación. Se encarga de manejar la navegación entre métodos, pasos y estados, además de dar acceso a los ajustes, la gestión de copias de seguridad y el resto de actividades. Lleva asociado un *navigation drawer* que permite acceder directamente a: los diferentes

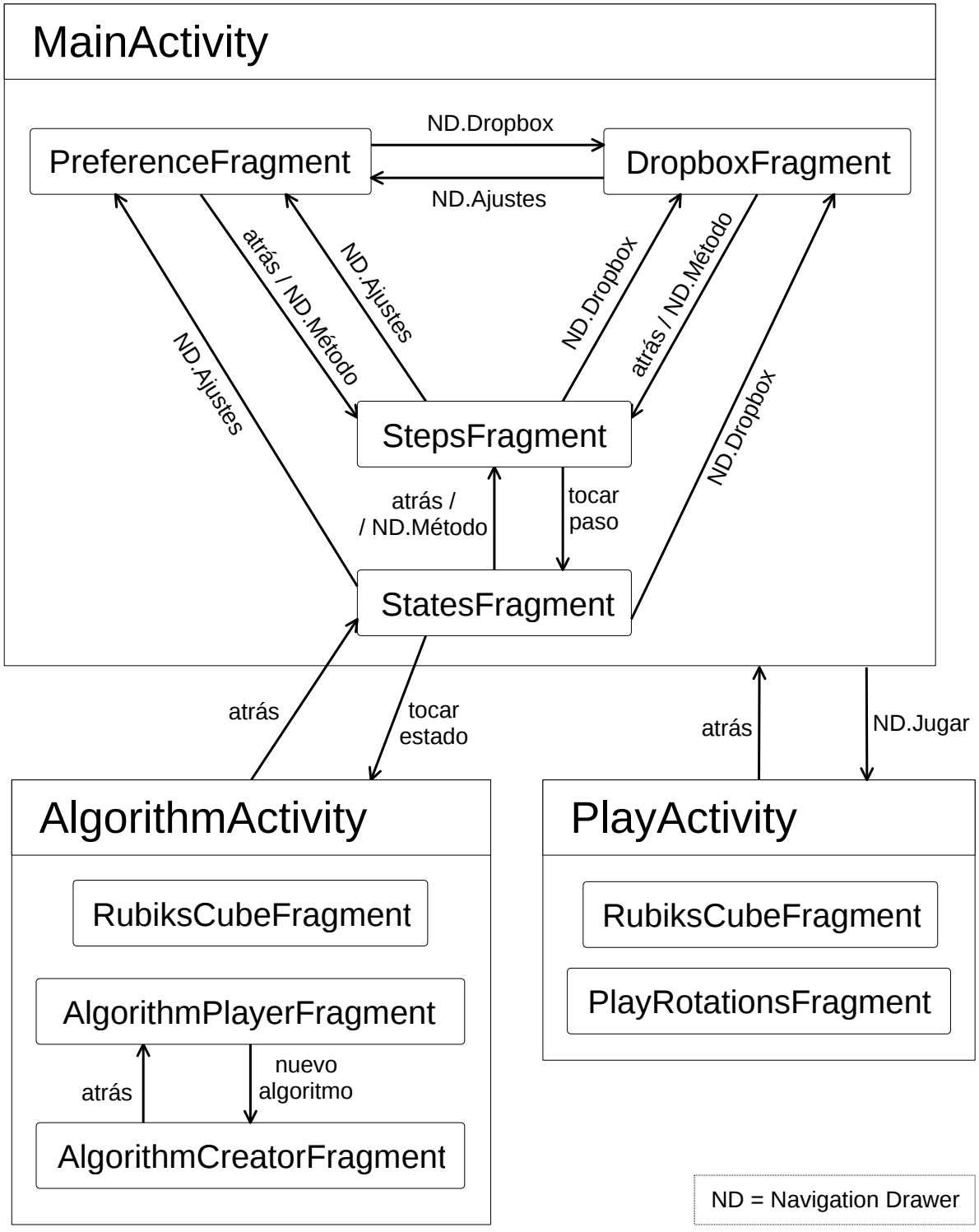

Figura 6.6: Navegación entre actividades y fragmentos

métodos, la actividad de juego libre con el cubo, el fragmento de gestión de copias de seguridad y la pantalla de ajustes. No importa qué fragmento de esta actividad se encuentra visible; el *navigation drawer* es inherente a ella y siempre puede ser mostrado deslizando el dedo desde el borde izquierdo de la pantalla del dispositivo.

- `AlgorithmsActivity`. Es la pantalla en la que el usuario puede visualizar en un cubo de Rubik tridimensional la ejecución de los algoritmos asociados a un determinado estado. En ella el usuario también puede crear algoritmos y guardarlos en la base de datos.

- `PlayActivity`. Propociona al usuario una pantalla en la que poder ejecutar rotaciones a placer sobre un cubo de Rubik inicialmente resuelto.

En el apéndice A se pueden ver capturas de todas las pantallas.

6.4.2. API de Dropbox

Dropbox proporciona una API, llamada «Core API» que permite leer y escribir archivos de su servicio sin tener que emplear sus aplicaciones oficiales o su página web. Además, para Android ha publicado un SDK que facilita la programación, dado que incluye métodos que envuelven las llamadas HTML a su API [27].

La aplicación permite al usuario conectarla con su cuenta de Dropbox. Para ello se usa el método `startOAuth2Authentication` de la clase `AndroidAuthSession` que, si el usuario tiene instalada la aplicación oficial de Dropbox en su dispositivo, la ejecutará para solicitar al usuario permisos de lectura y escritura de archivos de su cuenta y, si no la tiene instalada, hará lo mismo a través del navegador.

Cuando el usuario termina de dar los permisos, la aplicación guarda en sus preferencias internas el token de acceso que devuelve el método `getOAuth2AccessToken`. Este token evita que el usuario necesite volver a dar acceso a la aplicación. Para ello, se recoge de las preferencias el token guardado y se le asigna a la sesión a través del método `setOAuth2AccessToken`. Posteriormente, con `isLinked` se asegura que la aplicación siga registrada en la cuenta de Dropbox del usuario.

Los métodos `getFile` y `putFile` permiten, respectivamente, descargar y subir archivos de Dropbox. Son usados, por tanto, para subir una copia de seguridad y descargar una ya existente en la cuenta del usuario. A la hora de subir, el nombre del archivo contendrá la hora y fecha actuales. De esta manera se consigue habilitar la existencia de múltiples copias de seguridad. A través del método `metadata`, la aplicación recoge las copias de seguridad existentes en la cuenta del usuario. Las enumera en pantalla mostrando la fecha y hora de cada una para que así el usuario pueda elegir una de ellas para descargar y aplicar.

Las copias de seguridad consisten en archivos JSON que guardan todos los algoritmos de la base de datos. Se vuelcan solamente los algoritmos porque son los únicos datos que

el usuario puede modificar y querer compartir con otras personas. La estructura de estas copias de seguridad es la siguiente:

```
[
  {
    "id": <number>,
    "fav_alg": <string>,
    "algs": [<string>, [...]]
  },
  [...]
]
```

7
Pruebas

Para verificar y validar la aplicación, es decir, para, respectivamente, comprobar que su implementación es correcta y que se cumplen los requisitos funcionales y no funcionales, se han llevado a cabo diferentes tipos de pruebas: inspección de código, pruebas unitarias de caja negra, pruebas sobre la interfaz de usuario y pruebas de usabilidad.

7.1. Inspección del código

Dado el carácter iterativo del proyecto, la inspección de código ha sido una constante durante el desarrollo del mismo.

Las múltiples refactorizaciones han servido para mejorar la estructura interna del proyecto, para organizar los paquetes de tal manera que se adecuasen a la evolución de éste. Además, se han visto acompañadas de inspección de código, cuyos resultados han sido:

- Mejoras en el nombrado de clases, métodos y variables para que el código sea más fácil de leer y mantener.
- Aumento del control de errores para dar mayor robustez a la aplicación.
- Reestructuración de métodos en aras de una mayor calidad del código.
- Corrección de errores.

7.2. Pruebas unitarias de caja negra

Las pruebas unitarias llevadas a cabo han sido de tipo caja negra. Éstas se caracterizan por verificar que las salidas de un método son las esperadas para las entradas probadas y por ignorar cómo se han implementado tales métodos.

Las pruebas se han realizado en el propio IDE de desarrollo Android Studio, gracias al soporte que éste da para ejecutar pruebas unitarias JUnit. Se han probado tanto parámetros correctos como incorrectos. De esta manera, se ha verificado que los métodos devolvían los resultados esperados y que lanzaban las excepciones adecuadas en caso de error.

A continuación se enumeran los métodos que se han probado, que han sido los de las clases de tipo `util`:

- Clase `StringsUtils`.
  - Método `getDefaultJsonBackupNameWithDate`. Se ha probado que el `String` devuelto contenga una fecha y una hora.
  - Método `getDateFromDefaultJsonBackupNameWithDate`. Se ha llamado con nombres válidos para verificar que devuelve la fecha y hora correctas, y nombres incorrectos para comprobar que se lanza la excepción esperada.
- Clase `CodificationUtil`.
  - Método `createCodificationToPaint`. Se ha probado enviarle todos los tipos de codificaciones posibles (`FULL`, `F2L`, `OLL` y `PLL`), tanto con codificaciones correctas para verificar que devuelve la correspondiente codificación entera del cubo de Rubik, como con codificaciones incorrectas para comprobar que se lanza la excepción esperada.
- Clase `ColorsUtils`. Se han probado los tres siguientes métodos para comprobar que la traducción de color HTML a RGB con valor entre 0 y 1 es correcta, y para verificar que ante un `String` que no represente un color HTML se lanza la excepción esperada:
  - Método `getRgbRedFromHtml`.
  - Método `getRgbGreenFromHtml`.
  - Método `getRgbBlueFromHtml`.

7.3. Pruebas sobre la interfaz de usuario

La interfaz de usuario se ha probado tanto para verificar el correcto funcionamiento de la aplicación como para validarla. Las pruebas se han llevado a cabo en un Nexus 5 [17] y han sido las siguientes:

- Pruebas de la correcta navegación entre pantallas. Han consistido en ejecutar las acciones de navegación indicadas en la figura 6.6 y verificar que las pantallas mostradas son las adecuadas.

- Pruebas de cada pantalla de manera individual. Han demostrado el correcto funcionamiento de la reproducción de algoritmos en el cubo de Rubik tridimensional, la conexión con Dropbox y la creación de copias de seguridad.

- Pruebas de la repercusión de las acciones realizadas en una pantalla sobre otras pantallas. Han verificado la siguiente funcionalidad: la creación y modificación de algoritmos, el cambio entre etiquetar un estado como aprendido o no, la elección de un tema de colores del cubo de Rubik y la descarga y volcado de copias de seguridad.

- Pruebas de cumplimiento de los estándares de Material design. Se ha usado la aplicación Keyline Pushing [18] para verificar las métricas recomendadas por Google.

7.4. Pruebas de usabilidad

Las pruebas de usabilidad se han llevado a cabo en cada iteración con el objetivo de determinar cómo de intuitiva era la interfaz.

Han consistido en observar a diferentes personas utilizar la aplicación sin haberles proporcionado previamente explicación alguna. Durante las pruebas, se anotaba en qué momentos los usuarios dudaban. Tras terminar de probar la aplicación, se les preguntaba en qué habían tenido dificultades, cómo pensaban que se podrían arreglar y qué mejoras sugerían.

Se ha observado que las personas que tenían experiencia con el cubo de Rubik manejaban la aplicación con soltura, mientras que aquellas que no la tenían no entendían muy bien todo lo que la aplicación ofrece. En particular, no sabían cuál era la diferencia entre los distintos métodos.

Toda la información recopilada ha permitido mejorar la interfaz gráfica. Los cambios más destacados han sido los siguientes:

- Se sustituyeron los iconos de los botones de ejecución de giro siguiente y anterior y del botón de deshacer todos los giros por otros más intuitivos.

- Se cambió el texto del botón de ajuste de la velocidad de giro del cubo de Rubik tridimensional. En concreto, se pasó de la notación «x2» a la notación «>>». Parte de los usuarios entendía que el botón aumentaba y disminuía un hipotético zoom del cubo.

- En la pantalla de juego libre con el cubo de Rubik, hay un botón que deshace el cubo ejecutando varias rotaciones aletoriamente. Se cambió el texto de este botón por la confusión que provocaba; se sustituyó «ALEATORIO» por «DESHACER».

Parte de los usuarios entendía que al presionar el botón, la acción que se iba a llevar a cabo era ejecutar al azar un único giro.

- Al deshacer el cubo a través del botón «DESHACER», las rotaciones se ejecutan con velocidad máxima; sin embargo, el botón de velocidad de giro no cambiaba. Algunos usuarios entendían que las rotaciones iban a realizarse a la velocidad que mostraba el botón de velocidad, en lugar de a velocidad máxima. Se optó entonces por desactivar —solamente durante la breve ejecución de los giros aleatorios— el botón de velocidad y cambiar su texto a aquel que indica velocidad máxima.

- El botón de descarga de copias de seguridad de Dropbox y posterior volcado de las mismas tenía el siguiente texto: «Eliminar los algoritmos de la aplicación y crearlos a partir de la copia de seguridad de Dropbox seleccionada». A simple vista, por comenzar con la palabra «Eliminar», algún usuario entendía que servía para eliminar la copia de seguridad seleccionada. Por tanto, se cambió el texto a «Descargar copia de seguridad de Dropbox y aplicarla». Además, se incorporó un diálogo de confirmación en el que se avisa al usuario de que los algoritmos que tiene en la aplicación van a ser eliminados si continúa con la operación.

- En la pantalla de selección del algoritmo favorito para resolver un estado, algunos usuarios no entendían que el algoritmo que encabezaba la pantalla era el actual favorito. Se incluyó, por tanto, una etiqueta de texto que lo indicase.

8 Mantenimiento

La aplicación será publicada en Google Play, plataforma desde la cual los usuarios la podrán instalar en sus dispositivos. A partir de entonces, la aplicación entrará en la fase de mantenimiento, que incluirá: mantenimiento perfectivo, mantenimiento preventivo estructural y mantenimiento correctivo.

8.1. Mantenimiento perfectivo

El mantenimiento perfectivo será necesario por las siguientes razones:

- Los usuarios querrán que la aplicación incorpore funcionalidad nueva que les sea útil.
- Android es un sistema operativo en constante evolución. La aplicación se ajustará a las nuevas versiones y evolucionará junto a los estándares de diseño.
- Los dispositivos Android son muy diferentes: desde móviles con pantallas de 3.5" a tabletas de 12". Será necesario ajustar la aplicación a la mayor cantidad de tamaños de pantalla y resoluciones posible.
- Actualmente existen varios métodos de resolución que la aplicación no incorpora. Éstos, junto a los nuevos que irán apareciendo en un futuro, deberán ser añadidos.

8.2. Mantenimiento preventivo estructural

El mantenimiento preventivo estructural se llevará a cabo sobre todo por los cambios que se realicen en el código derivados del mantenimiento perfectivo. Incluirá:

- Mejorar la documentación existente.
- Reestructurar el código para mejorar la legibilidad.
- Aumentar los tests.

8.3. Mantenimiento correctivo

Con el mantenimiento correctivo se diagnostican errores y se corrigen. Será de dos tipos:

- De emergencia ante los errores que encuentren los usuarios que utilicen la aplicación.
- Planificado para tener en cuenta los fallos que se puedan cometer con la inclusión de futura funcionalidad.

9

Conclusiones

El presente Trabajo de Fin de Grado ha permitido la construcción de una aplicación Android destinada a que la gente aprenda a resolver el cubo de Rubik de diferentes maneras a través de una interfaz atractiva. Ha supuesto investigar las soluciones ya existentes, encontrar las limitaciones de éstas y crear un proyecto de software que satisfaga las necesidades de las personas interesadas en este rompecabezas.

El proyecto ha seguido un ciclo de vida iterativo incremental en el que se han sucedido las fases de educción de requisitos, diseño, implementación y pruebas, hasta llegar a conseguir una aplicación similar a la idea inicial que se tenía de ella. Actualmente se encuentra en fase de mantenimiento.

La aplicación se ha construido con el lenguaje Java empleando el SDK de Android. Para el modelo se ha hecho uso de SQLite para construir la base de datos y de archivos en formato JSON para importar y exportar datos. La vista se ha elaborado a partir de archivos XML que definen los componentes gráficos estáticos y de código en Java que crea otros dinámicamente. Además, el cubo de Rubik tridimensional se ha pintado utilizando la tecnología OpenGL ES. El controlador se basa en clases `Activity` y `Fragment` que construyen las diferentes pantallas de la aplicación y manejan la navegación entre ellas, además de en métodos proporcionados por el SDK de Dropbox para contactar con su servicio.

Durante el transcurso del Trabajo de Fin de Grado se han podido poner en práctica los conocimientos adquiridos en la carrera. *Ingeniería del Software* ha aportado saber cómo gestionar un proyecto de software; *Desarrollo de Aplicaciones para Dispositivos Móviles*, una mayor práctica en el desarrollo de aplicaciones para Android; *Álgebra Lineal y Geometría*, entender mejor los principios de OpenGL.

La experiencia de la elaboración del Trabajo de Fin de Grado ha sido bastante

enriquecedora. Por una parte, se ha construido desde cero un proyecto de una envergadura no despreciable. El paso por sus distintas fases y la visión de su evolución ha sido algo realmente valioso. Durante la implementación se ha aprendido a utilizar OpenGL ES para construir objetos tridimensionales. Además, se han asentado y mejorado los conocimientos sobre el desarrollo de aplicaciones Android y el uso de pautas de diseño gráfico para las aplicaciones de este sistema operativo. Por otra parte, se ha aprendido a utilizar varias herramientas y se ha mejorado el uso de otras, como por ejemplo: GitHub [28], Android Studio [6], Sublime Text [29], LaTeX [30], Cacoo [21], Dia [20], Gimp [31] e Inkscape [32].

10

Líneas de trabajo futuro

El presente Trabajo de Fin de Grado ha permitido desarrollar una aplicación para dispositivos Android que facilita el aprendizaje de métodos de resolución del cubo de Rubik. Además de cumplir con los objetivos iniciales, se ha añadido funcionalidad adicional para mejorar la aplicación. No obstante, es posible aportar más calidad a través de una serie de mejoras que se describen a continuación.

Adaptación a los diferentes dispositivos Android

La interfaz gráfica de la aplicación se ha diseñado tomando como referencia un Nexus 5, cuya pantalla tiene un tamaño de 4.95" y una resolución Full HD (1920 x 1080). Ocurre que los dispositivos Android son muy variados y sus pantallas muy distintas. Aunque parte de las medidas asignadas a los componentes gráficos son relativas, otras muchas son absolutas, y deben ser adaptadas a esta variedad de pantallas. Por otra parte, para las tabletas, cuyas pantallas son comparativamente muy grandes, no basta con solamente modificar las dimensiones; es necesario también modificar la disposición de las pantallas para aprovechar el espacio disponible. El uso de fragmentos en el controlador facilitará esta adaptación.

Adición de nuevos métodos de resolución

Aunque en un principio se pensó que la aplicación incluyese únicamente el método de resolución CFOP, el diseño y la implementación han permitido que la adición de otros métodos sea una tarea fácil. Además del método de principiantes y del método a ciegas 3OP, incluidos ambos en la aplicación, hay muchos otros que se irán añadiendo

progresivamente, comenzando con aquellos más usados.

Posibilidad de creación de temas de colores

La aplicación permite, a través de los ajustes, elegir el tema de colores que se usa para pintar el cubo de Rubik tridimensional. Sin embargo, estos temas vienen predefinidos. Una buena idea sería incorporar un sistema a través del cual el usuario pudiese elegir cualquier color para crear temas personalizados.

Mejorar la usabilidad

La idea en el inicio del Trabajo de Fin de Grado era que la aplicación fuese empleada por aquellas personas conocedoras de los métodos de resolución rápida. Al final se incluyó, entre otros, un método para principiantes. Dado que así el público objetivo se amplía, es necesario que la aplicación sea cómoda y sencilla también para aquellos que quieren aprender a resolver el cubo de Rubik. Sin embargo, actualmente no lo es, pues las pruebas de usabilidad mostraron que los principiantes no entendían muy bien todo lo que la aplicación ofrece. Como era de esperar, no conocían los métodos avanzados y no sabían qué diferencia había entre ellos.

Por lo tanto, existe una necesidad de mejorar la usabilidad de la aplicación. Diferentes maneras de conseguirlo son:

- Incluir una sección de ayuda que explique por qué existen diferentes métodos de resolución. Hacer que se muestre la primera vez que la aplicación se ejecute.
- Añadir explicaciones de los pasos en el método para iniciados.
- Incluir Google Analytics [33] para recoger datos sobre cómo usan la aplicación los diferentes grupos de usuarios que la instalen.

Atender las peticiones de los usuarios

La publicación en Google Play de la aplicación permitirá que una gran cantidad de usuarios la instale en sus dispositivos. A través de esta plataforma, el comportamiento esperado es que los usuarios la valoren y escriban qué desean que incluya. Además, se anunciará la existencia de la aplicación en aquellos foros en los que se reúne la mayoría de los entusiastas del cubo de Rubik. Las opiniones recibidas se tendrán en cuenta para la continua mejora de la aplicación.

Bibliografía

[24] Google. *Material design*. [Último acceso: 01/07/2015]. URL: https://www.google.com/design/spec/material-design/introduction.html.

[25] Google. *Android Developers - Develop Apps*. [Último acceso: 01/07/2015]. URL: https://developer.android.com/intl/es/develop/index.html.

[26] Kevin Brothaler. *OpenGL ES 2.0 for Android*. Ed. por Susannah Davidson Pfalzer. The Pragmatic Programmers, 2013. ISBN: 978-1-937785-34-5.

Referencias

Las siguientes direcciones web enlazan a herramientas, bibliotecas, aplicaciones e información adicional mencionadas en la memoria.

[1] *WCA (World Cube Associations)*. [Último acceso: 01/07/2015]. URL: https://www.worldcubeassociation.org/.

[2] *Applet de Werner Randelshofer*. [Último acceso: 01/07/2015]. URL: http://www.randelshofer.ch/cube/rubik/?MR2MU2MF2.

[3] *Cubo de Rubik Algoritmos y Más*. [Último acceso: 01/07/2015]. URL: https://play.google.com/store/apps/details?id=com.jqrapps.cubealgorithms.

[4] *Rubik's Cube Fridrich Solver*. [Último acceso: 01/07/2015]. URL: https://play.google.com/store/apps/details?id=com.BodorGdev.fridrichsolver2.

[5] *Rubik's Cube OLL/PLL Trainer*. [Último acceso: 01/07/2015]. URL: https://play.google.com/store/apps/details?id=de.dennishafemann.rubikscube3x3x3ollplltrainer.

[6] *Android Studio*. [Último acceso: 01/07/2015]. URL: https://developer.android.com/sdk/index.html.

[7] *IntelliJ IDEA Community Edition*. [Último acceso: 01/07/2015]. URL: https://www.jetbrains.com/idea/.

[8] *Android NDK*. [Último acceso: 01/07/2015]. URL: https://developer.android.com/tools/sdk/ndk/index.html.

[9] *AutoFitTextView*. [Último acceso: 01/07/2015]. URL: https://github.com/grantland/android-autofittextview.

[10] *CircularProgressBar*. [Último acceso: 01/07/2015]. URL: https://github.com/castorflex/SmoothProgressBar.

[11] *FButton*. [Último acceso: 01/07/2015]. URL: https://github.com/hoang8f/android-flat-button.

[12] *FloatingActionButton*. [Último acceso: 01/07/2015]. URL: https://github.com/makovkastar/FloatingActionButton.

[13] *Material Dialogs*. [Último acceso: 01/07/2015]. URL: https://github.com/afollestad/material-dialogs.

[14] *Picasso*. [Último acceso: 01/07/2015]. URL: https://github.com/square/picasso.

[15] *SmartTabLayout*. [Último acceso: 01/07/2015]. URL: https://github.com/ogaclejapan/SmartTabLayout.

[16] *Gama de dispositivos Nexus*. [Último acceso: 01/07/2015]. URL: http://www.google.com/nexus/.

[17] *Nexus 5*. [Último acceso: 01/07/2015]. URL: http://www.google.com/intl/ALL/nexus/5/.

[18] *Keyline Pushing*. [Último acceso: 01/07/2015]. URL: https://play.google.com/store/apps/details?id=com.faizmalkani.keylines.

[19] *GitHub - Student Developer Pack*. [Último acceso: 01/07/2015]. URL: https://education.github.com/pack.

[20] *Dia*. [Último acceso: 01/07/2015]. URL: https://wiki.gnome.org/Apps/Dia/.

[21] *Cacoo*. [Último acceso: 01/07/2015]. URL: https://cacoo.com/.

[22] *Dropbox - Core API para Android*. [Último acceso: 01/07/2015]. URL: https://www.dropbox.com/developers/core/sdks/android.

[23] *Google I/O 2014 - Material design*. [Último acceso: 01/07/2015]. URL: https://www.google.com/events/io/io14videos/79edef8b-96d4-e311-b297-00155d5066d7.

[27] *Dropbox - Core API*. [Último acceso: 01/07/2015]. URL: https://www.dropboxstatic.com/static/developers/dropbox-android-sdk-1.6.3-docs/index.html.

[28] *GitHub*. [Último acceso: 01/07/2015]. URL: https://github.com/.

[29] *Sublime Text*. [Último acceso: 01/07/2015]. URL: https://www.sublimetext.com/.

[30] *LaTeX*. [Último acceso: 01/07/2015]. URL: http://www.latex-project.org/.

[31] *Gimp*. [Último acceso: 01/07/2015]. URL: http://www.gimp.org/.

[32] *Inkscape*. [Último acceso: 01/07/2015]. URL: https://inkscape.org/.

[33] *Google Analytics SDK for Android*. [Último acceso: 01/07/2015]. URL: https://developers.google.com/analytics/devguides/collection/android/.

Apéndices

A

Manual de usuario

Al ejecutar la aplicación por vez primera, ésta mostrará una pantalla con los pasos del método para principiantes (ver figura A.1). Si se quiere cambiar a otro método hay que abrir el *navigation drawer* (ver figura A.2); para ello, basta con deslizar el dedo desde la parte izquierda de la pantalla o tocar el icono que tiene tres líneas horizontales y que se encuentra en la barra superior.

En la lista de pasos de un método es posible tocar uno de ellos cualquiera para ver los estados posibles de dicho paso. Si tocamos en el paso «Last layer - Permutation of the corners», accederemos a la pantalla A.3. En esta pantalla, el icono con las tres líneas horizontales ha sido sustituido por una flecha. Ahora, al pulsarla, no se abrirá el *navigation drawer*, sino que se volverá a la pantalla inicial. No obstante, sigue siendo posible abrirlo deslizando el dedo desde la parte izquierda de la pantalla (ver figura A.4).

Si seleccionamos, por ejemplo, el primer estado, accederemos a la pantalla de la figura A.5. En ella aparece un cubo de Rubik tridimensional capaz de reproducir las rotaciones que componen el algoritmo que se muestra. Para ejecutarlas una tras otra automáticamente, basta dar al icono *play*. Si se quiere reproducir únicamente la siguiente o la anterior, entonces hay que usar las flechas *adelante* y *atrás*. El botón situado a la derecha permite deshacer todas las rotaciones ejecutadas. Por otra parte, el botón situado al lado del cubo, con el texto «>>>>>», sirve para incrementar y disminuir la velocidad de giro.

El usuario puede indicar que ha aprendido a resolver el estado a través del *switch* situado en la parte inferior (ver figura A.6). Si volvemos a la pantalla anterior, el estado «Case 1» estará marcado como aprendido (figura A.7). En la pestaña «ALL» aparecen todos los estados del paso, mientras que en la pestaña «LEARNED» solamente los aprendidos (figura A.8) y en la pestaña «NOT LEARNED» los no aprendidos (figura A.9).

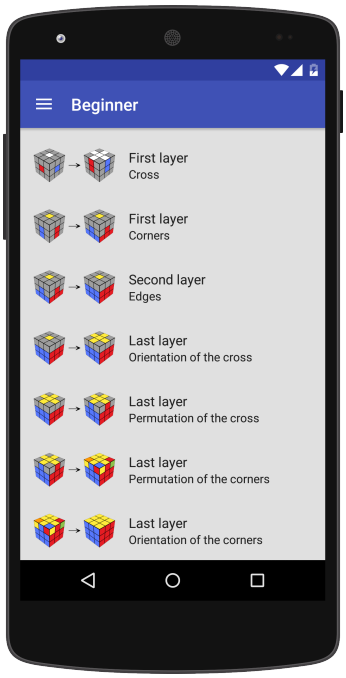

Figura A.1: Pantalla inicial de la aplicación

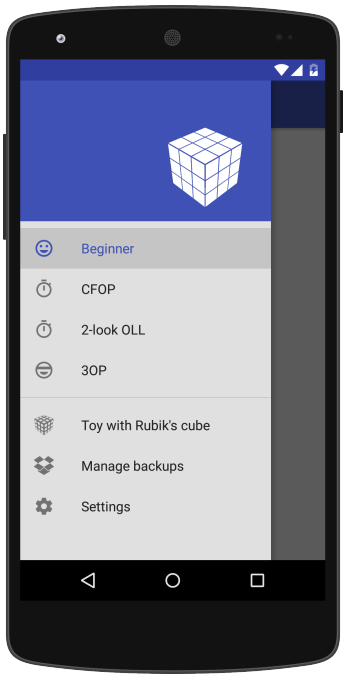

Figura A.2: *Navigation drawer* abierto desde la pantalla inicial

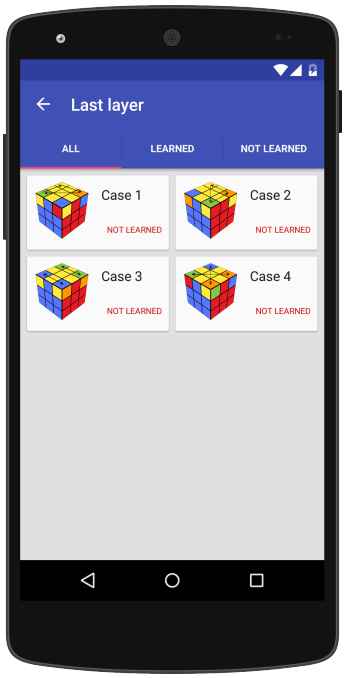

Figura A.3: Pantalla con los estados del paso «Last layer - Permutation of the corners»

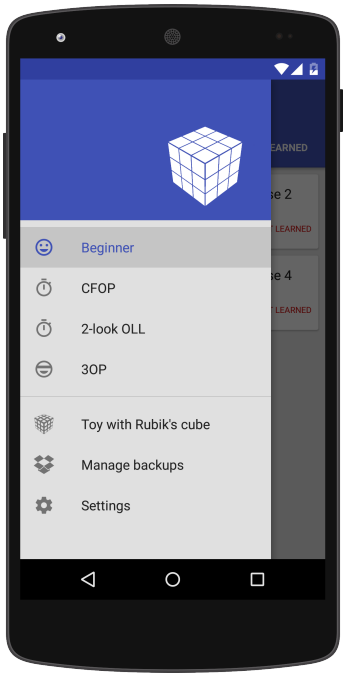

Figura A.4: *Navigation drawer* abierto desde la pantalla de los estados de un paso

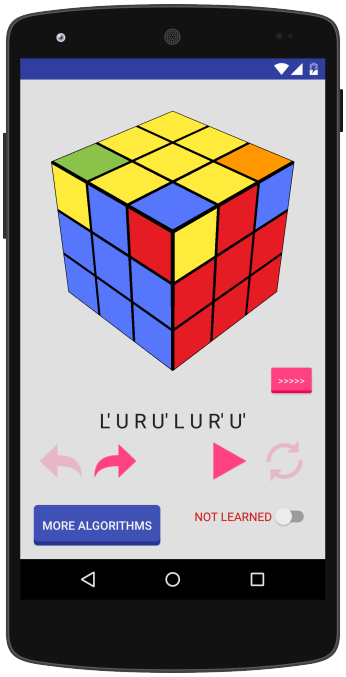

Figura A.5: Pantalla de reproducción de algoritmos. El usuario no ha aprendido el estado.

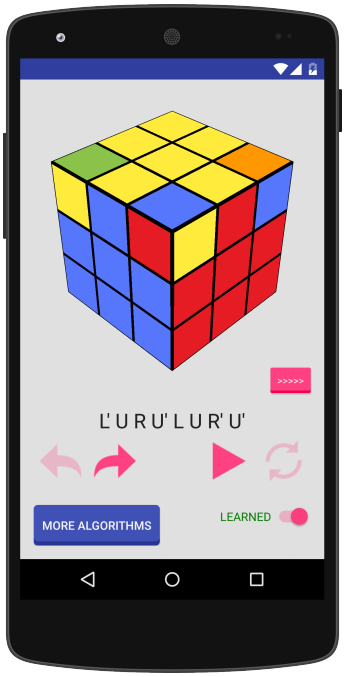

Figura A.6: Pantalla de reproducción de algoritmos. El usuario ha aprendido el estado.

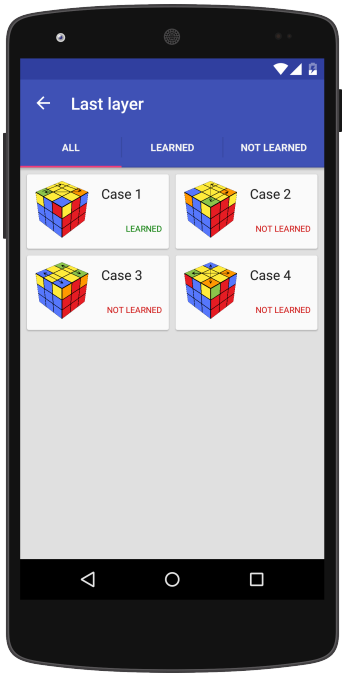

Figura A.7: Pantalla de estados. La pestaña «ALL» muestra todos.

Figura A.8: Pantalla de estados. La pestaña «LEARNED» muestra los que el usuario ha aprendido.

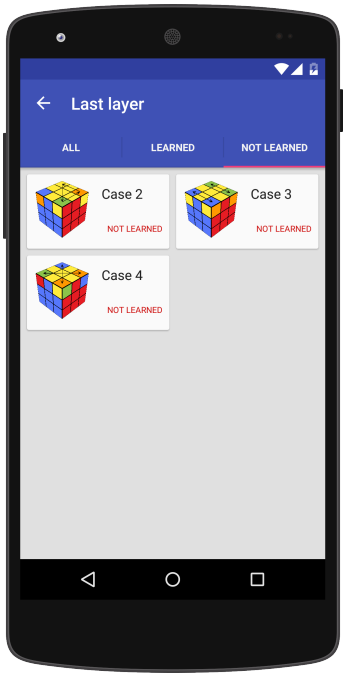

Figura A.9: Pantalla de estados. La pestaña «NOT LEARNED» muestra los que el usuario no ha aprendido.

Creación de algoritmos

Para añadir o eliminar algoritmos de un estado, hay que pulsar el botón «MORE ALGORITHMS» (ver figura A.10). Se mostrará cuál es el algoritmo favorito (que es el que aparece en la pantalla de reproducción) y una lista con el resto de algoritmos del estado. En la figura A.11 vemos que no hay ninguno además del favorito. Para crear uno nuevo pulsamos el botón con el signo «+», que lleva a la pantalla de creación de algoritmos (ver figura A.12).

Para crear algoritmos es necesario pulsar los botones que representan cada una de las 36 rotaciones posibles (ver apéndice B). Usar o no paréntesis no afecta a cómo el cubo de Rubik vaya a reproducir el giro; solamente son usados para facilitar la lectura de los algoritmos. No obstante, es necesario que el balance de paréntesis sea correcto, pues en caso contrario no se podrá guardar el algoritmo creado (ver figuras A.13 y A.14). Ante error, se puede pulsar el botón deshacer para eliminar tanto los paréntesis como las rotaciones. Tras guardar el algoritmo, un snackbar aparecerá en la parte inferior para informar de que la operación se ha llevado a cabo satisfactoriamente (ver figura A.15).

Vuelta a la lista de algoritmos del estado, ahora aparece el nuevo que hemos creado (ver figura A.16). Si lo tocamos, entonces pasará a ser el algoritmo favorito con el que resolver el estado (ver figura A.17). Si deseamos eliminar un algoritmo, basta con mantenerlo pulsado. Un diálogo se mostrará al usuario para que confirme su acción (ver figura A.18).

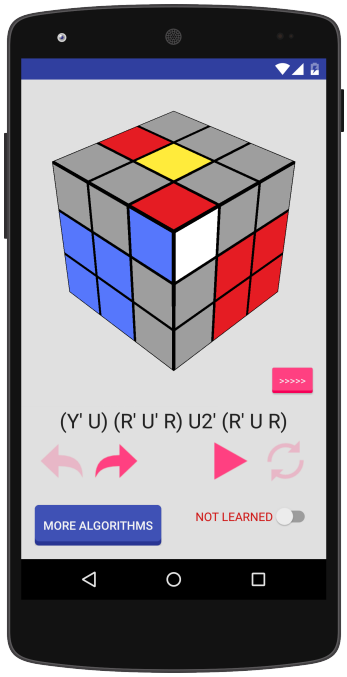

Figura A.10: El botón «MORE ALGORITHMS» muestra la lista de algoritmos

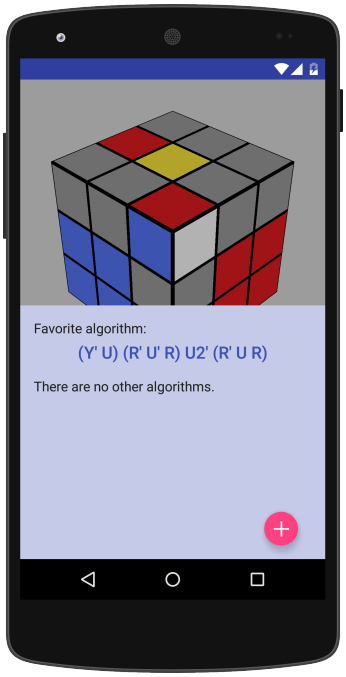

Figura A.11: Lista de algoritmos. Solamente un algoritmo resuelve el estado

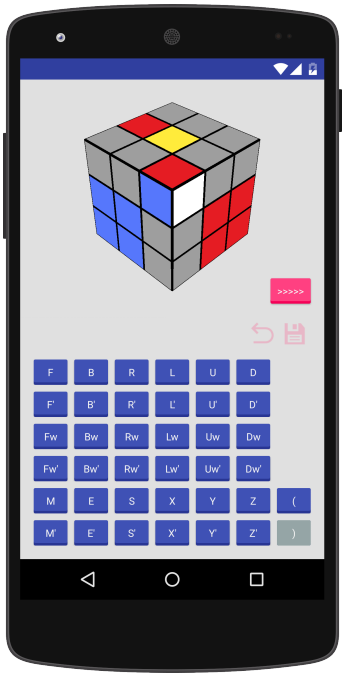

Figura A.12: Pantalla de creación de algoritmos

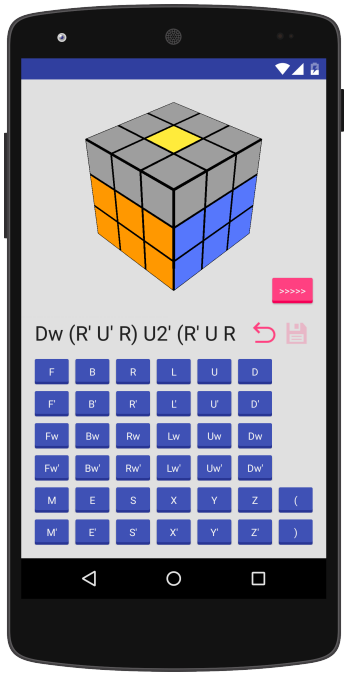

Figura A.13: El balance de paréntesis es incorrecto y no se puede guardar el algoritmo

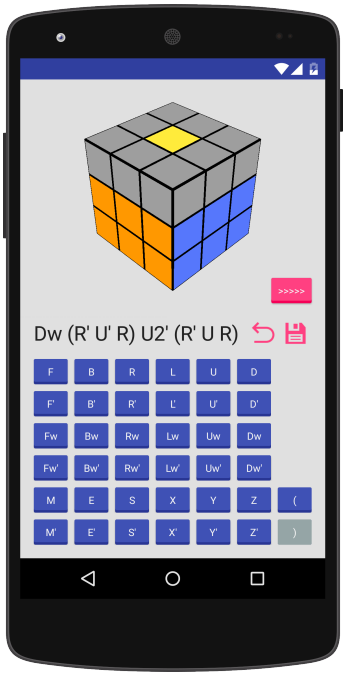

Figura A.14: El balance de paréntesis es correcto y el algoritmo se puede guardar

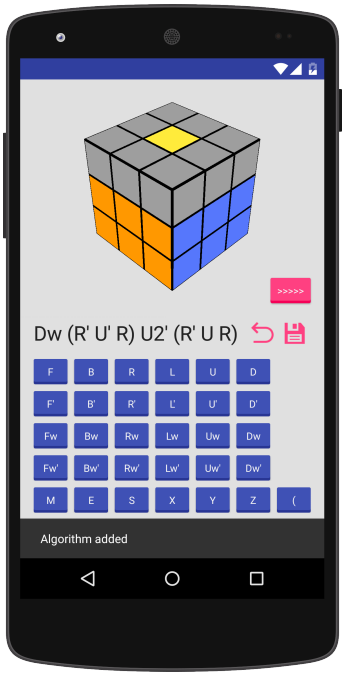

Figura A.15: Algoritmo correctamente guardado

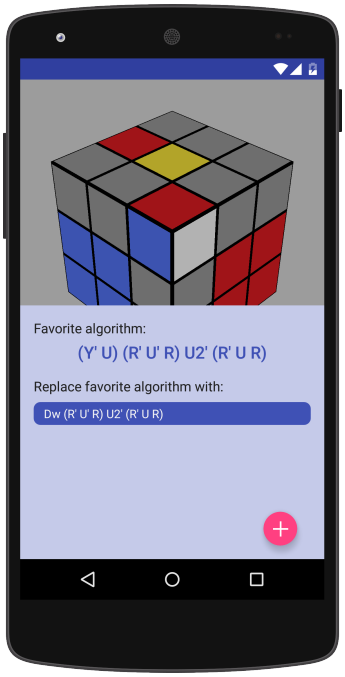

Figura A.16: La lista de algoritmos ahora muestro el nuevo creado

Figura A.17: Tras elegir el nuevo algoritmo, éste se muestra en la pantalla de reproducción

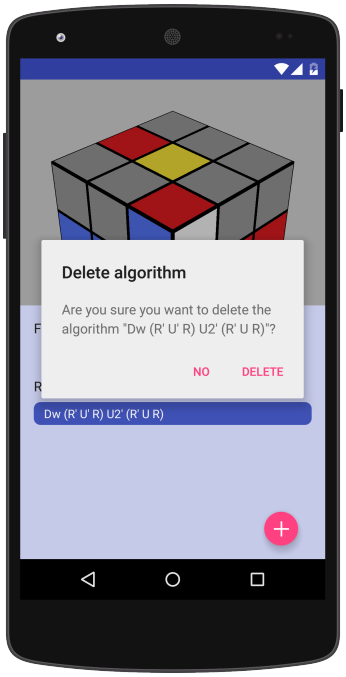

Figura A.18: Un diálogo de confirmación avisa al usuario de que va a eliminar un algoritmo

Dropbox

Si el usuario tiene cuenta de Dropbox, puede conectar la aplicación a ella. De esta manera, podrá realizar copias de seguridad de todos los algoritmos que contenga la aplicación. Para ello debe dirigirse, a través del *navigation drawer*, a la pantalla de gestión de copias de seguridad en Dropbox. Si el dispositivo no está conectado a internet, la aplicación avisará de ello (ver figura A.19) y solamente permitirá volver a comprobar la conexión. Si se tiene conexión, se mostrará la pantalla de la figura A.20. En ese caso será posible crear nuevas copias de seguridad de los algoritmos que contenga la aplicación y subirlas a Dropbox (ver figura A.21). Éstas se mostrarán en una lista y serán descritas con la fecha y la hora en la que fueron creadas.

Dado que es posible tener múltiples copias de seguridad, el botón de descarga no se habilitará hasta que se seleccione una, como se muestra en la figura A.22. Dado que al aplicar una copia de seguridad se borran todos los algoritmos de la aplicación para reemplazarlos con los existentes en la copia, se muestra un diálogo de confirmación al usuario para que sea consciente de ello (ver figura A.23).

Durante todas las acciones de esta pantalla, se mostrará la pantalla A.24 mientras se espera la respuesta de los servidores de Dropbox.

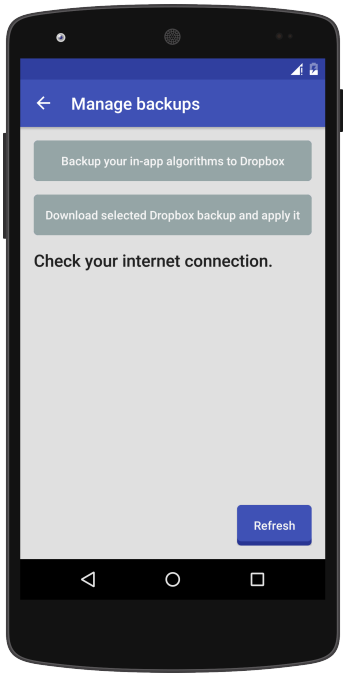

Figura A.19: Se avisa de que no hay conexión y se deshabilitan los botones

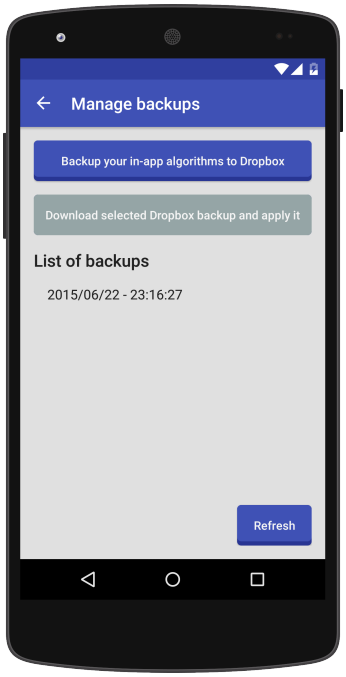

Figura A.20: Pantalla de gestión de copias de seguridad en Dropbox

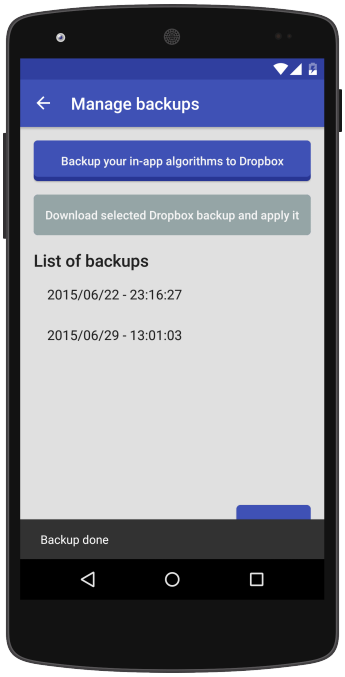

Figura A.21: Un snackbar informa de que la copia de seguridad se ha subido a Dropbox correctamente

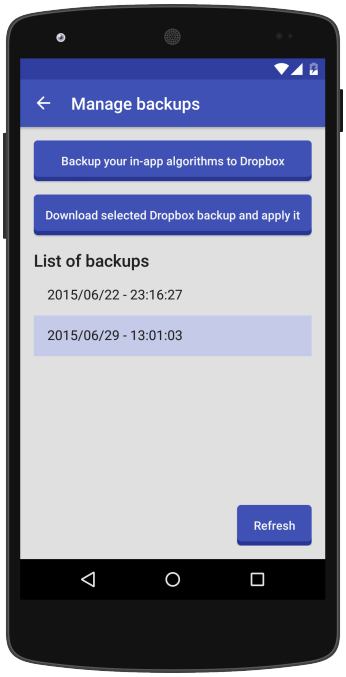

Figura A.22: El botón de descarga de copia de seguridad se habilita tras seleccionar una de la lista

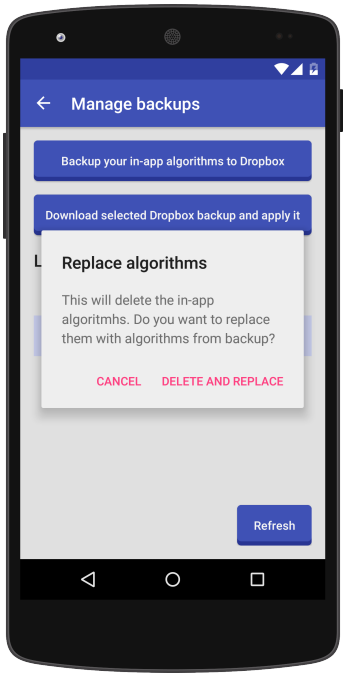

Figura A.23: Antes de descargar una copia de seguridad, el usuario debe confirmar la acción

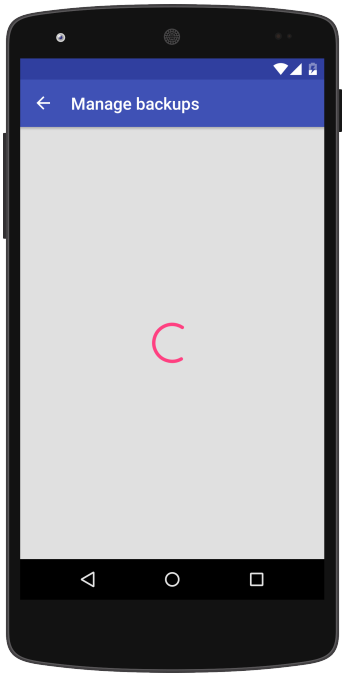

Figura A.24: Pantalla de espera indeterminada

Ajustes

Desde el *navigation drawer* es posible acceder a los ajustes (ver figura A.25). Allí se podrán ver detalles del autor de la aplicación, de las bibliotecas de código libre de terceros y se podrá escoger el tema de colores con el que pintar el cubo de Rubik tridimensional (ver figura A.26).

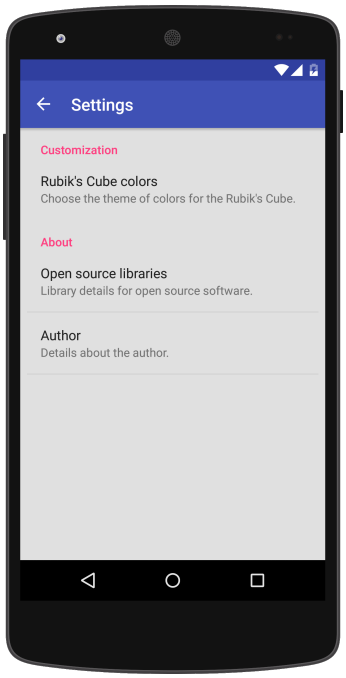

Figura A.25: Pantalla de ajustes

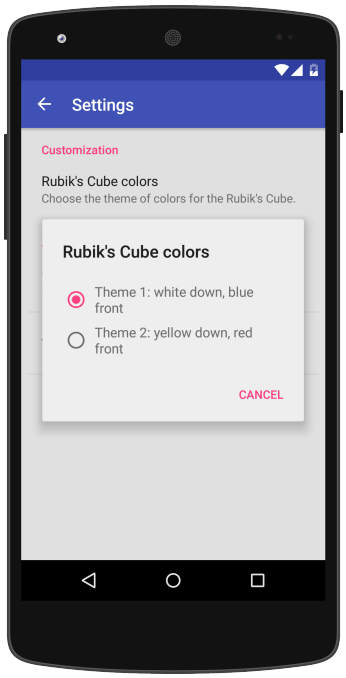

Figura A.26: Desde la pantalla de ajustes se puede elegir el tema de colores

Juego libre

La aplicación incorpora una pantalla para que el usuario pueda mover libremente el cubo de Rubik (ver figura A.27). A ella se llega desde el *navigation drawer*. Contiene los botones que ya se vieron en la pantalla de creación de algoritmos, necesarios para ejecutar cualquier rotación. Además, proporciona un botón «SCRAMBLE» que permite deshacer el cubo de Rubik aleatoriamente (ver figura A.28). Como se muestra en la figura A.29, mientras se deshace el cubo —lo cual es casi inmediato— todos los botones se desactivan.

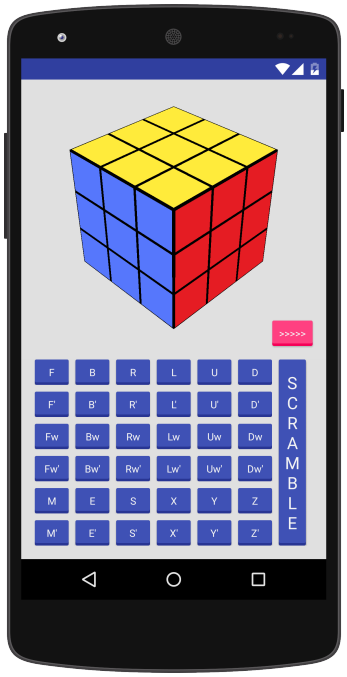

Figura A.27: Pantalla de juego libre con el cubo de Rubik

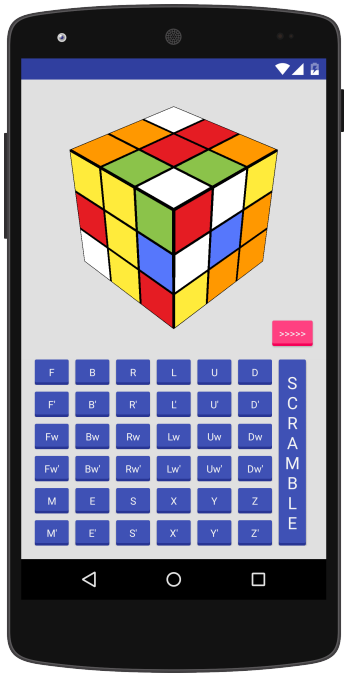

Figura A.28: Cubo de Rubik deshecho tras pulsar el botón «SCRAMBLE»

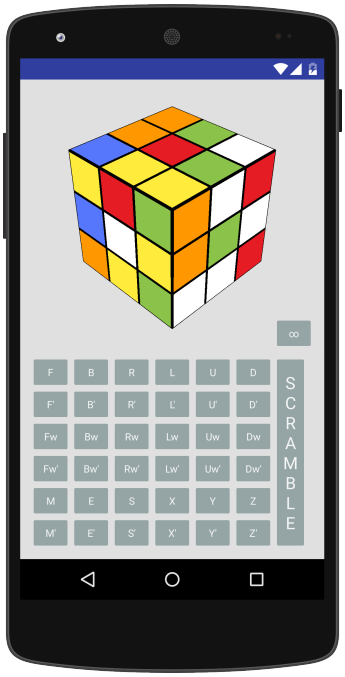

Figura A.29: Mientras se deshace el cubo no se puede tocar ningún botón

B

Rotaciones del cubo de Rubik

En primer lugar es necesario saber diferenciar una cara de una capa:

- Una cara es un conjunto formado por la pegatina de un centro y las ocho que la rodean. El cubo de Rubik tiene 6 caras, una por cada centro.
- Una capa es un conjunto formado por:
  - o bien cuatro cubitos esquina, cuatro cubitos arista y un centro,
  - o bien cuatro cubitos esquina y cuatro centros.

  El cubo de Rubik tiene 9 capas: las 6 correspondientes a cada cara y 3 intermedias.

Las caras pueden girarse en sentido horario (ver figuras B.1, B.3, B.5, B.7, B.9, B.11) o en sentido antihorario (ver figuras B.2, B.4, B.6, B.8, B.10, B.12).

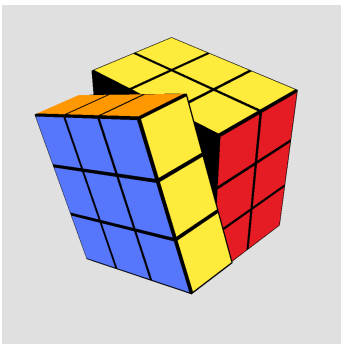

Figura B.1: Giro horario de la cara frontal

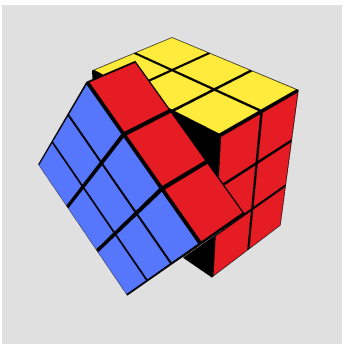

Figura B.2: Giro antihorario de la cara frontal

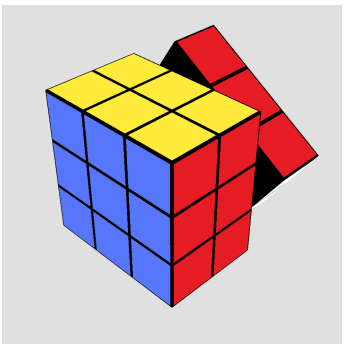

Figura B.3: Giro horario de la cara trasera

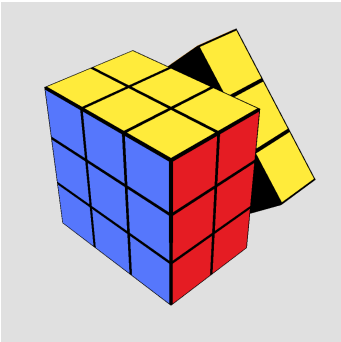

Figura B.4: Giro antihorario de la cara trasera

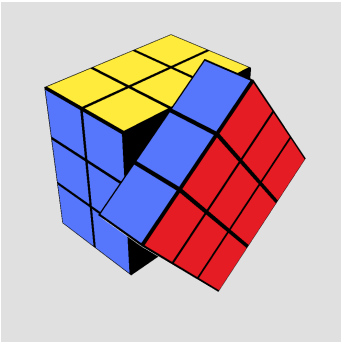

Figura B.5: Giro horario de la cara derecha

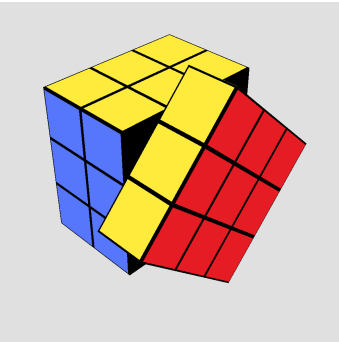

Figura B.6: Giro antihorario de la cara derecha

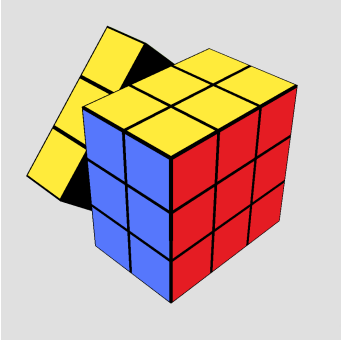

Figura B.7: Giro horario de la cara izquierda

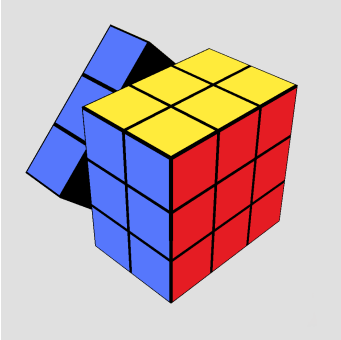

Figura B.8: Giro antihorario de la cara izquierda

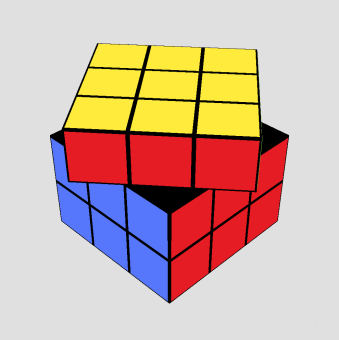

Figura B.9: Giro horario de la cara superior

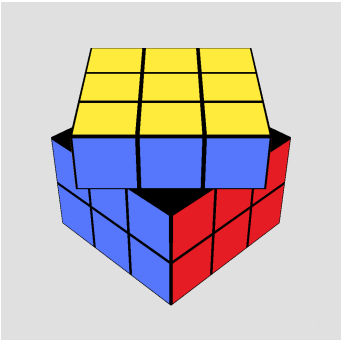

Figura B.10: Giro antihorario de la cara superior

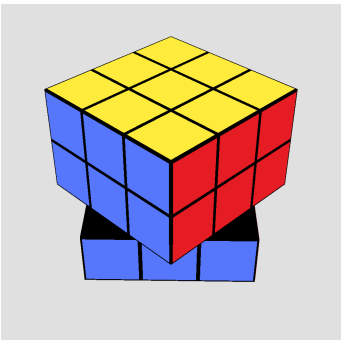

Figura B.11: Giro horario de la cara inferior

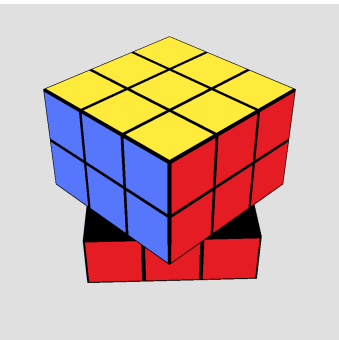

Figura B.12: Giro antihorario de la cara inferior

Además de las seis caras, también es posible girar las tres capas intermedias. Y de nuevo en sentido horario o antihorario, como se ve en las figuras B.13, B.14, B.15, B.16, B.17, B.18.

Otras doce rotaciones son aquellas que giran una determinada cara y la capa intermedia adherida a ella, ya sea en sentido horario (ver figuras B.19, B.21, B.23, B.25, B.27, B.29) o en sentido antihorario (ver figuras B.20, B.22, B.24, B.26, B.28, B.30).

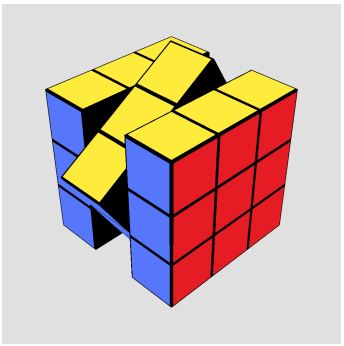

Figura B.13: Giro horario de la capa situada entre la izquierda y la derecha

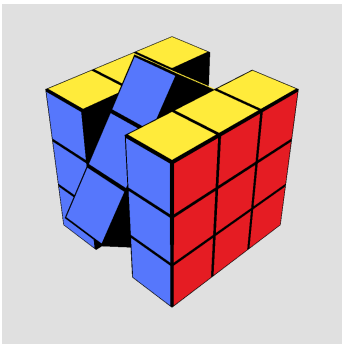

Figura B.14: Giro antihorario de la capa situada entre la izquierda y la derecha

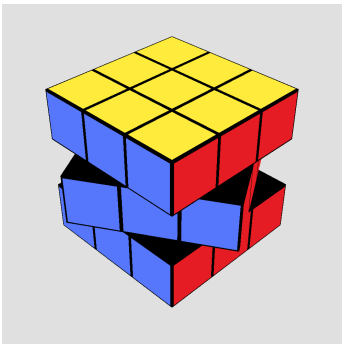

Figura B.15: Giro horario de la capa situada entre la inferior y la superior

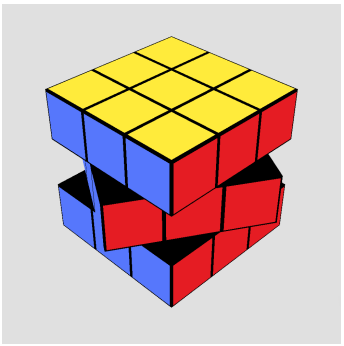

Figura B.16: Giro antihorario de la capa situada entre la inferior y la superior

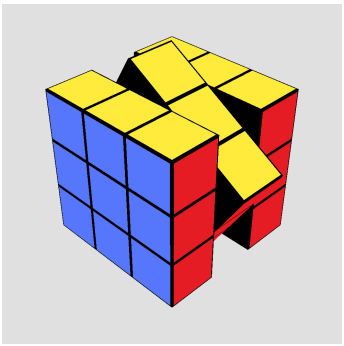

Figura B.17: Giro horario de la capa situada entre la frontal y la trasera

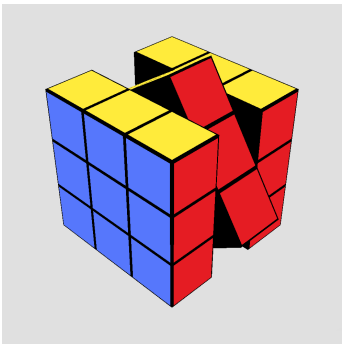

Figura B.18: Giro antihorario de la capa situada entre la frontal y la trasera

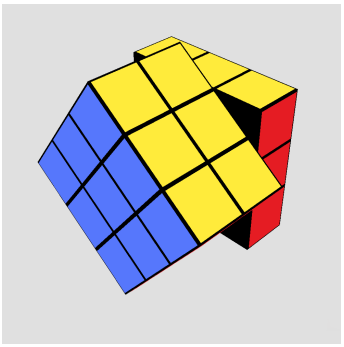

Figura B.19: Giro horario de la cara frontal y la capa intermedia adherida a ella

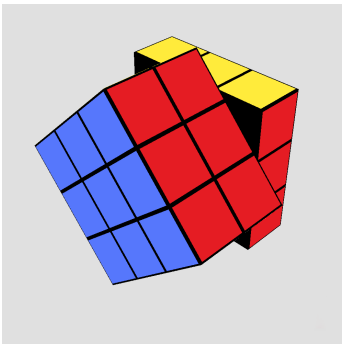

Figura B.20: Giro antihorario de la cara frontal y la capa intermedia adherida a ella

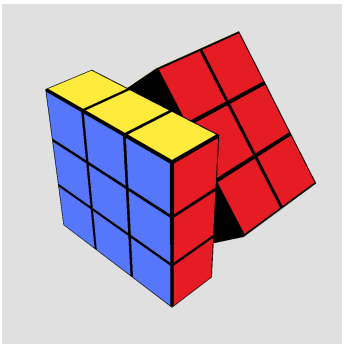

Figura B.21: Giro horario de la cara trasera y la capa intermedia adherida a ella

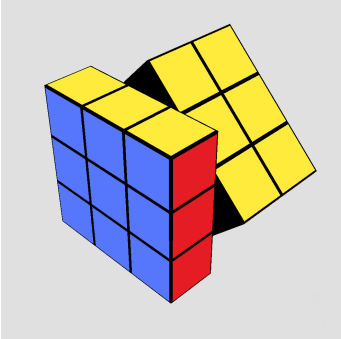

Figura B.22: Giro antihorario de la cara trasera y la capa intermedia adherida a ella

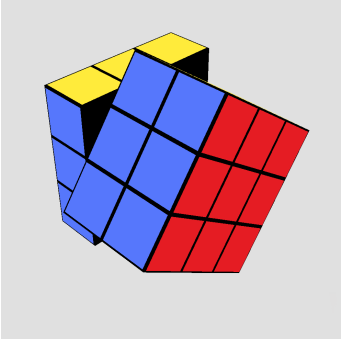

Figura B.23: Giro horario de la cara derecha y la capa intermedia adherida a ella

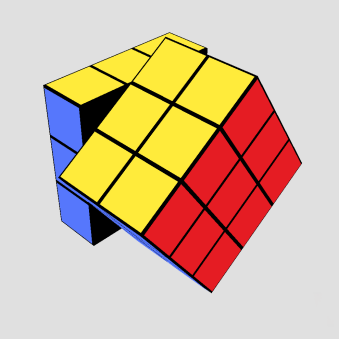

Figura B.24: Giro antihorario de la cara derecha y la capa intermedia adherida a ella

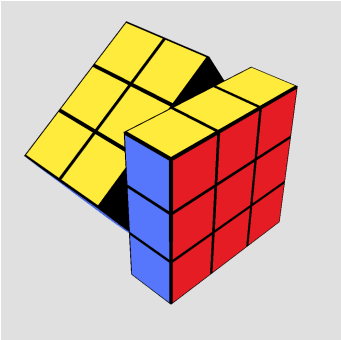

Figura B.25: Giro horario de la cara izquierda y la capa intermedia adherida a ella

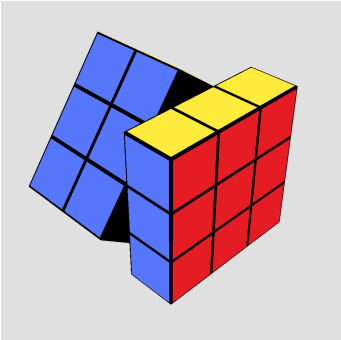

Figura B.26: Giro antihorario de la cara izquierda y la capa intermedia adherida a ella

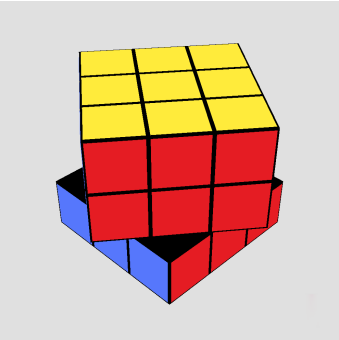

Figura B.27: Giro horario de la cara superior y la capa intermedia adherida a ella

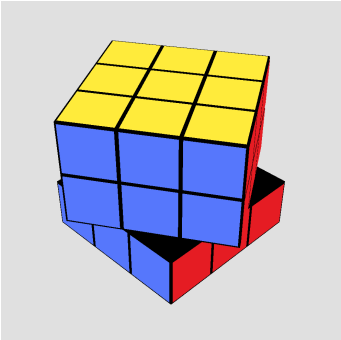

Figura B.28: Giro antihorario de la cara superior y la capa intermedia adherida a ella

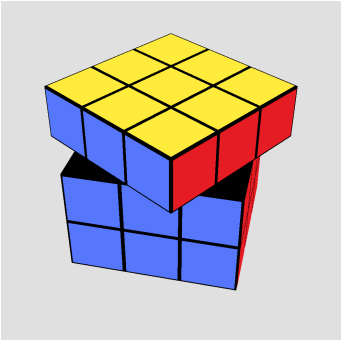

Figura B.29: Giro horario de la cara inferior y la capa intermedia adherida a ella

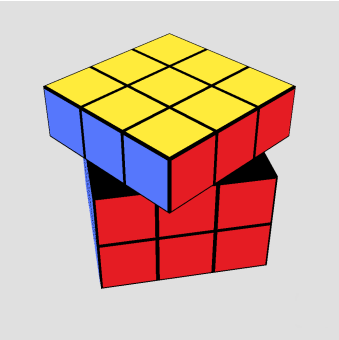

Figura B.30: Giro antihorario de la cara inferior y la capa intermedia adherida a ella

Los últimos seis giros corresponden a aquellos que rotan el cubo de Rubik completamente respecto a los ejes X, Y, Z (X el horizontal, Y el vertical y Z el perpendicular a estos dos) en sentido horario y antihorario, como se ve en las figuras B.31, B.32, B.33, B.34, B.35, B.36.

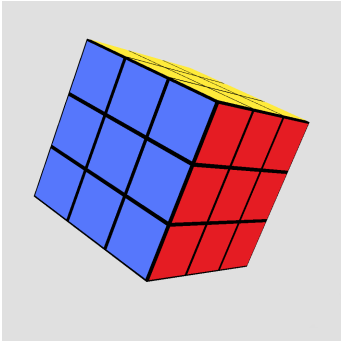

Figura B.31: Giro horario del cubo en torno al eje X

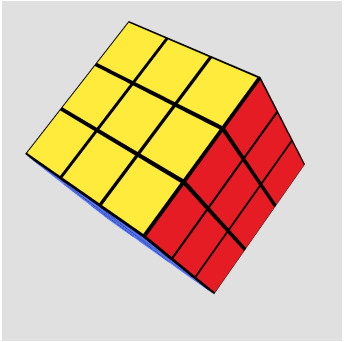

Figura B.32: Giro antihorario del cubo en torno al eje X

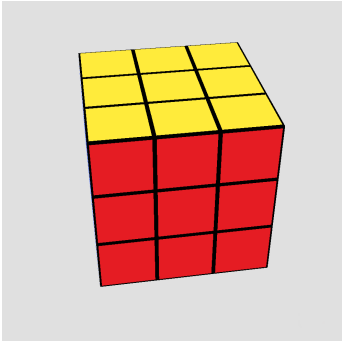

Figura B.33: Giro horario del cubo en torno al eje Y

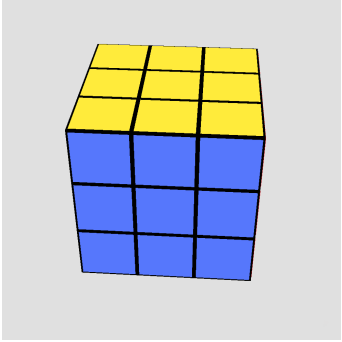

Figura B.34: Giro antihorario del cubo en torno al eje Y

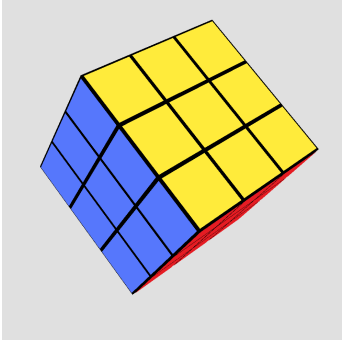

Figura B.35: Giro horario del cubo en torno al eje Z

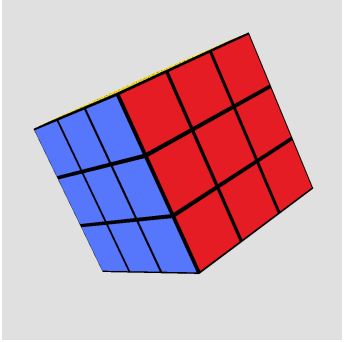

Figura B.36: Giro antihorario del cubo en torno al eje Z

En la subsección 6.2.2 se habló del enumerado `RotationType`. Los valores que puede tomar son aquellos que representan las 36 diferentes rotaciones que admite el cubo de Rubik. Son:

- `FRONT_CLOCKWISE`, `BACK_CLOCKWISE`, `RIGHT_CLOCKWISE`, `LEFT_CLOCKWISE`, `TOP_CLOCKWISE` y `BOTTOM_CLOCKWISE` para las rotaciones en sentido horario de las caras.
- `FRONT_ANTICLOCKWISE`, `BACK_ANTICLOCKWISE`, `RIGHT_ANTICLOCKWISE`, `LEFT_ANTICLOCKWISE`, `TOP_ANTICLOCKWISE` y `BOTTOM_ANTICLOCKWISE` para las rotaciones en sentido antihorario de las caras.
- `MIDDLE_CLOCKWISE`, `EQUATOR_CLOCKWISE` y `STANDING_CLOCKWISE` para las rotaciones en sentido horario de las capas intermedias.

- `MIDDLE_ANTICLOCKWISE`, `EQUATOR_ANTICLOCKWISE` y `STANDING_ANTICLOCKWISE` para las rotaciones en sentido antihorario de las capas intermedias.
- `FRONT_W_CLOCKWISE`, `BACK_W_CLOCKWISE`, `RIGHT_W_CLOCKWISE`, `LEFT_W_CLOCKWISE`, `TOP_W_CLOCKWISE` y `BOTTOM_W_CLOCKWISE` para las rotaciones en sentido horario de cada cara y la capa intermedia adherida a ella.
- `FRONT_W_ANTICLOCKWISE`, `BACK_W_ANTICLOCKWISE`, `RIGHT_W_ANTICLOCKWISE`, `LEFT_W_ANTICLOCKWISE`, `TOP_W_ANTICLOCKWISE` y `BOTTOM_W_ANTICLOCKWISE` para las rotaciones en sentido antihorario de cada cara y la capa intermedia adherida a ella.
- `X_CLOCKWISE`, `Y_CLOCKWISE` y `Z_CLOCKWISE` para las rotaciones en sentido horario del cubo en torno a los ejes X, Y o X.
- `X_ANTICLOCKWISE`, `Y_ANTICLOCKWISE` y `Z_ANTICLOCKWISE` para las rotaciones en sentido antihorario del cubo en torno a los ejes X, Y o X.

C

Codificaciones de los estados

El color de las pegatinas es lo que diferencia un estado de un cubo de Rubik de otro. A la hora de dibujar el cubo tridimensional con OpenGL ES, este color viene dado por la codificación y el tipo de codificación del estado (atributos `codification` y `codification_type` de la tabla `state`, sección 5.2).

Para poder pintar el cubo, es necesario crear previamente una cadena de caracteres de la que se pueda interpretar de qué color se pintará cada pegatina. Esta cadena estará formada por los números de los colores existentes en la base de datos (atributo `number` de la tabla `color`, sección 5.2). Los dígitos que la componen son los siguientes:

- Los 24 primeros representan los colores de las 3 pegatinas de cada una de las 8 esquinas.
- Los 24 siguientes representan los colores de las 2 pegatinas de cada una de las 12 aristas.
- Los 6 últimos representan los colores de la pegatina de cada uno de los 6 centros.

A modo de ejemplo, la codificación de un cubo de Rubik resuelto sería `146136145135-246236245235161413154636453526242325123456`. Dependiendo del número atribuido a cada color en la base de datos y del tema de colores escogido, el cubo resuelto tendrá unos colores u otros.

Hay que recordar que la definición de los diferentes estados que van a aparecer en la aplicación se realiza en un archivo JSON. Este archivo será leído la primera vez que se instale la aplicación para guardar su información en la base de datos. Cualquier estado puede ser añadido en el archivo JSON con una codificación de 54 caracteres como la que

se acaba de describir. No obstante, para simplificar la labor de escritura de codificaciones, se han creado tipos que aprovechan que cierto conjunto de pegatinas van a tener siempre un determinado color. Estos tipos los recoge el enumerado `CodificationType` (ver subsección 6.2.2). Sus valores son: `FULL`, `F2L`, `OLL` y `PLL`. Mientras que con el tipo `FULL` hay que indicar los colores de las 54 pegatinas, el resto de tipos se aprovechan de lo siguiente:

1. `F2L`. Las dos primeras capas (ver apéndice B para entender la distinción entre «cara» y «capa») están resueltas, excepto el par esquina-arista que debe ser colocado en estas dos primeras capas, en las caras frontal y derecha. Además, las esquinas y las aristas de la capa superior son irrelevantes, por lo que esta capa se pintará entera del color neutro —gris en las figuras abajo mostradas—, excepto si la esquina o la arista a colocar están en ella. Ver figura C.1.
2. `OLL`. Las dos primeras capas están resueltas. De la capa superior solamente se pintan las pegatinas de la cara superior; el resto tendrá el color neutro. Ver figura C.2.
3. `PLL`. Las dos primeras capas y la cara superior están resueltas. Ver figura C.3.

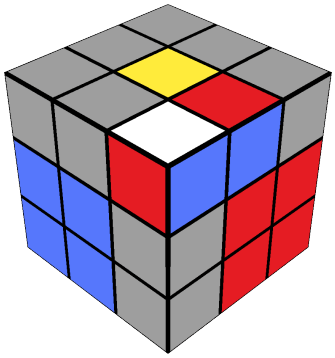

Figura C.1: Ejemplo de estado del paso F2L

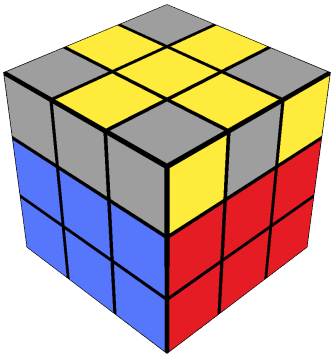

Figura C.2: Ejemplo de estado del paso OLL

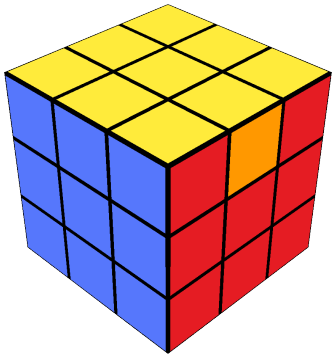

Figura C.3: Ejemplo de estado del paso PLL

Para la codificación de tipo `F2L` solamente es necesario indicar en qué posiciones se encuentran la esquina y la arista y qué colores tienen sus pegatinas. En la figura C.4 se muestran los números que hay escribir en la codificación para indicar estas posiciones. Así, la codificación del estado de la figura C.1 sería `43160813`, que indicaría:

- La esquina está en la posición 4.
- Los colores de la esquina son 316.
- La arista está en la posición 8.
- Los colores de la arista son 13.

Para la codificación de tipo `OLL` es necesario indicar en qué posición se encuentran las pegatinas de la cara superior. Éstas pertenecerán a esquinas y aristas de la capa superior. El esquema para esta codificación se muestra en la figura C.5. Así, la codificación del estado de la figura C.2 sería `11111111`, que indicaría:

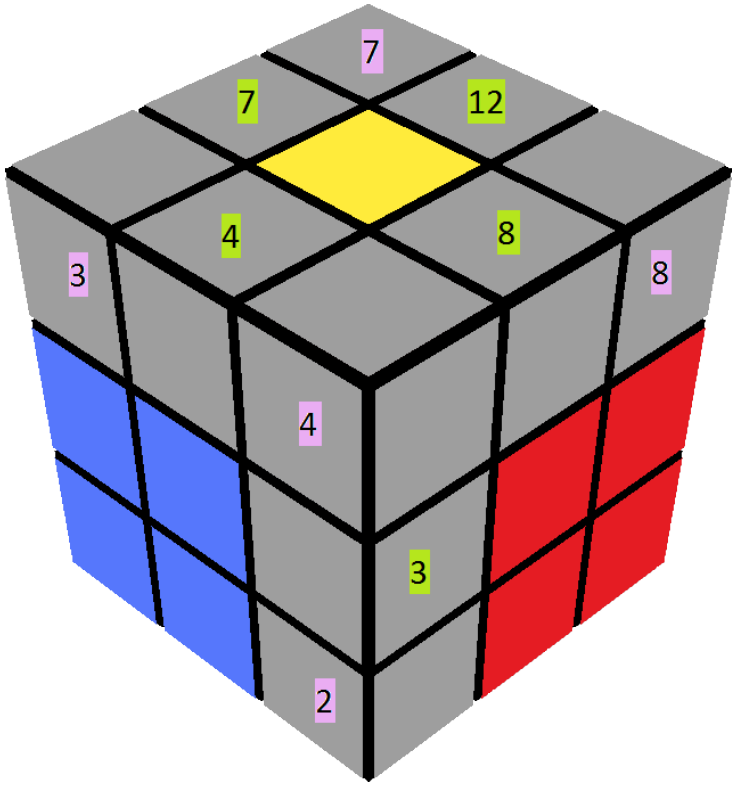

Figura C.4: Esquema para el tipo de codificación `F2L`

- La pegatina de la esquina inferior izquierda (según el esquema C.5) se encuentra en la posición 1 —y no en la 0 ni en la 2—.
- La pegatina de la esquina inferior derecha se encuentra en la posición 1.
- La pegatina de la esquina superior izquierda se encuentra en la posición 1.
- La pegatina de la esquina superior derecha se encuentra en la posición 1.
- La pegatina de la arista inferior se encuentra en la posición 1.
- La pegatina de la arista izquierda se encuentra en la posición 1.
- La pegatina de la arista derecha se encuentra en la posición 1.
- La pegatina de la arista superior se encuentra en la posición 1.

Para la codificación de tipo `PLL` solamente es necesario indicar los colores de las pegatinas de la cara superior que no se encuentran en la cara superior, como se muestra en la figura C.6. Así, la codificación del estado de la figura C.3 sería `141324231243`, que indicaría:

- La pegatina de la posición 1 tiene color 1.
- La pegatina de la posición 2 tiene color 4.

- La pegatina de la posición 3 tiene color 1.
- La pegatina de la posición 4 tiene color 3.
- La pegatina de la posición 5 tiene color 2.
- La pegatina de la posición 6 tiene color 4.
- La pegatina de la posición 7 tiene color 2.
- La pegatina de la posición 8 tiene color 3.
- La pegatina de la posición 9 tiene color 1.
- La pegatina de la posición 10 tiene color 2.
- La pegatina de la posición 11 tiene color 4.
- La pegatina de la posición 12 tiene color 3.

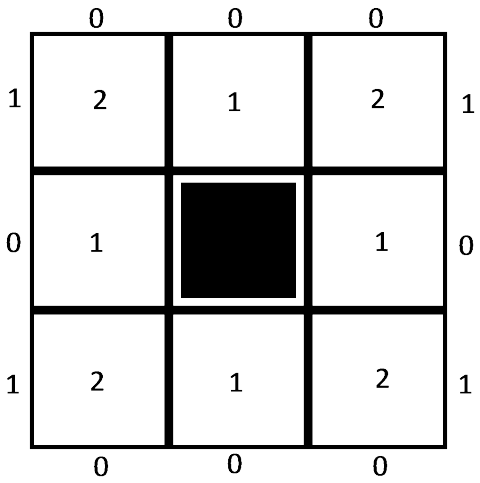

Figura C.5: Esquema para el tipo de codificación `OLL`

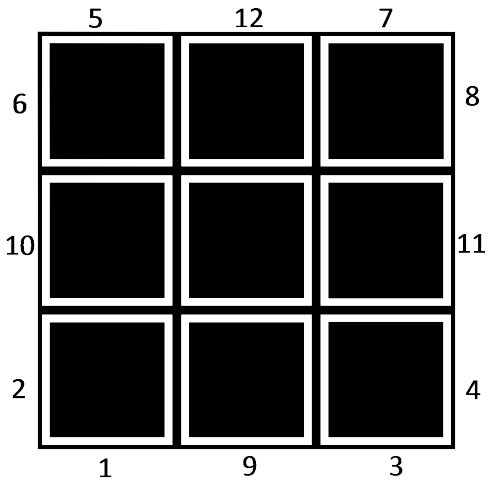

Figura C.6: Esquema para el tipo de codificación `PLL`